Schieb · Thrun · Wrobel

COMMODORE



intern

EIN DATA BECKER BUCH

DAS STEHT DRIN:

Ein Standardwerk zum COMMODORE 128, das für jeden unentbehrlich ist, der tiefer in den COMMODORE 128 einsteigen will. Das gesamte Betriebssystem ist ausführlich und gründlich kommentiert, Grafik, Soundbausteine, Prozessor und Peripherieanschlüsse sind genauestens beschrieben. Ein Buch, das für den professionellen Programmierer sehr schnell unentbehrlich wird.

Aus dem Inhalt:

- Der VIC-Chip
  Registerbelegung
  Betriebsarten
  Zeichendarstellung, Graphik, Sprites und Soft-Scrolling
- Ein- und Ausgabesteuerung
  Die CIAs im COMMODORE 128
  Echtzeituhr
  Der serielle IEC-Bus des COMMODORE 128
- Der Sound-Chip SID
- Der 8563-VDC-Chip
  Pinbelegung
  Nutzung der VDC-Register
  Hires-Graphik mit 640×200 Punkten
- Das Memory-Management, die MMU
- Assemblerprogrammierung (Nutzung der Kernal-Routinen)
- Einbinden neuer BASIC-Befehle
- BASIC-Tokens
- Die CPU 8502
- Zeilenweise dokumentiertes Kernal-ROM
- ausführlich dokumentiertes BASIC 7.0
- Z-80-ROM dokumentiert (Boot-Sektor)
- Betriebssystem und Monitorlisting
- Die Hardware

UND GESCHRIEBEN HABEN DIESES BUCH:

Engagierte Mitarbeiter des DATA BECKER-Teams. Frank Thrun, Programmierer in der Softwareabteilung, Jörg Schieb, erfahrener Maschinenspracheprogrammierer, und Heinz Wrobel kennen den COMMODORE 128 in- und auswendig.

ISBN 3-89011-098-3


Schieb · Thrun · Wrobel
COMMODORE
128
intern
EIN DATA BECKER BUCH

ISBN 3-89011-098-3

Wichtiger Hinweis:
Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technischen Angaben und Programme in diesem Buch wurden von dem Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

Inhaltsverzeichnis

1. Grundsätzliches zum C128

1.1 Einführung zum Commodore 128

Nach dem Commodore-Erfolgsschlager C64 folgten einige Versuche wie der Plus 4, C16 und C116, die der Markt zunächst nicht so recht annehmen wollte.

Diese Nachfolger des C64 boten aber nichts eigentlich Neues; ganz anders der Commodore 128. Er vereint praktisch drei Rechner in sich: Den wohlbekannten C64, für den ein Berg an Software und Literatur existiert, der bestimmt schon größer als der europäische Butterberg ist. Dann beinhaltet er, basierend auf den *Erfolgschips* 6510 (6502), VIC, SID, 6526 etc., den eigentlich *neuen* Rechner - den Commodore 128. Hier wurde er sinnvollerweise um einen 80-Zeichen-Video-Controller bereichert, der den C128 zur professionellen Maschine werden läßt. Man hat den VIC-Chip und die 6510 ein wenig verändert - jedoch haben sie nicht ihr eigentliches Gesicht verloren. Unverständlich ist vielleicht, warum man nicht als Mikroprozessor den 65C02 gewählt hat, der ebenfalls schneller arbeitet, zur 6502 kompatibel ist und noch einige nützliche Kommandos zusätzlich hinzubekommen hat. Dies hätte den C64-Modus überhaupt nicht beeinflußt. Man hat sich aber für den 8500 entschieden, der doppelt so schnell arbeiten kann wie sein Vorgänger, die 6510. Ferner ist der C128 auch ein CP/M-Rechner, der unter CP/M 3.0+ arbeitet. CP/M 3.0 ist die gängige Version für 128-KByte-Rechner. Hierzu ist noch ein Z80 eingebaut, der sogar mit 4 MHz gefahren wird. Allerdings wird die Geschwindigkeit gedämpft, wenn auf den Bus zugegriffen wird, da dieser nicht für diese Geschwindigkeit konzipiert worden ist.

Wir wollen uns mit den beiden Modi C64 und C128 gleichermaßen beschäftigen, da sie - das ist unsere persönliche Meinung - gleichberechtigt und ebenso gleichermaßen interessant sind. Schwerpunkt ist natürlich aber der C128-Modus, gerade was das ROM-Listing und die Zeropage betrifft. Einige Dinge lassen

sich aber besser im C64-Modus erläutern, beispielsweise der VIC-Chip, den wir im entsprechenden Kapitel noch genauer erläutern.

Das Buch *128 Intern* reiht sich in die Folge der INTERN-Werke von DATA BECKER ein und verfolgt das Ziel, dem Leser alle wichtigen Möglichkeiten des Rechners zu vermitteln. Deswegen wird auf jeden Baustein einzeln und ausführlich eingegangen, wobei der BASIC-Programmierer, ob Anfänger oder Profi, gleichermaßen tiefen Einblick gewinnen soll; der Assembler-Programmierer wird wohl den größten Nutzen davon tragen. Natürlich können nicht alle Möglichkeiten erschöpfend behandelt werden, auch soll dieses Buch keine BASIC-Einführung sein, jedoch werden die vorhandenen Möglichkeiten zumindest erwähnt. Die Realisierung obliegt dann dem Leser.

Im Commodore 128 befindet sich eine Vielzahl von modernen Bausteinen; die Platine ist reichlich bestückt. Um all dieses Innenleben koordinieren zu können, ist schon ein nicht unerheblicher hard- und softwaremäßiger Aufwand nötig. Dazu kommt noch das sehr komfortable und umfangreiche BASIC 7.0. Wichtige Eigenschaften des Commodore 128 sind die folgenden:

* *128 KByte dynamisches RAM*
* *2 mal 4 KByte Zeichengenerator*
* *Farbvideocontroller (VIC) mit Hi-Res-Grafik*
* *80-Zeichen-Videocontroller (VDC) mit RGB-Ausgang*
* *Auch auf dem 80-Zeichen-Monitor Hi-Res-Grafik*
* *Synthesizer mit drei unabhängigen Stimmen (polyphon)*
* *32 KByte BASIC-ROM!*
* *16 KByte Betriebsystem*
* *2 Parallel-I/Os*
* *2 Bildschirme gleichzeitig*

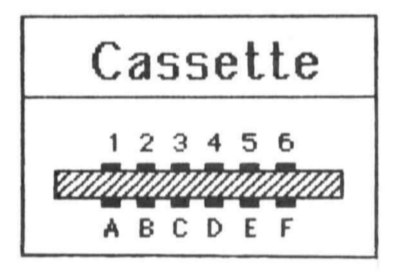
Wir wollen nun auf die verschiedenen Ein- und Ausgänge des Commodore 128 eingehen. Die Ausgänge für die Monitore werden dabei nicht berücksichtigt, da ihnen bzw. den Chips, die sie versorgen, je ein spezielles Kapitel gewidmet ist.

1.2 Der Datasetten-Anschluß

Der Commodore 128 wäre der erste Commodore-Rechner, der diesen Anschluß nicht hätte. Die Datasette hat zwar immer mehr an Bedeutung verloren, da die Flöppy-Disk-Stationen immer preiswerter geworden sind, jedoch hat alles mit dem legendären *PET* begonnen, der ja bekanntlich ein eingebautes Kassettengerät besessen hat - genau dies ist die Datasette. An diesenm Anschluß kann man ausschließlich Commodore-Kassettenlaufwerke anschließen; es sei denn, man ist der Bastelei verfallen. Ist dies vielleicht noch ein Grund zur Ärgernis, so muß man doch erwähnen, daß die Commodore-Kassettenlaufwerke seit je her eine sehr hohe Genauigkeit aufweisen, was man von anderen Rechnern zumindestens anfangs nicht behaupten konnte.

Die Datasette wird über ihren Anschluß sowohl mit Daten als auch mit ihrer Betriebsspannung versorgt. Die Daten flitzen bitseriell durchs Kabel (also wie beim IEC-Bus). Neben den Leitungen für Schreib- und Lesedaten gibt es noch eine Leitung, um den Laufwerksmotor ein- und auszuschalten, sowie eine Leitung, die dazu dient, die PLAY-Taste abzufragen.

Wie die einzelnen Leitungen genau belegt sind, können Sie den folgenden Abbildungen entnehmen:



PIN	SIGNAL
A-1	GND
B-2	+5V
C-3	CASSETTE MOTOR
D-4	CASSETTE READ
E-5	CASSETTE WRITE
F-6	CASSETTE SENSE

Abb. 1

1.3 Der Userport

Roboter durch den Commodore 128 gesteuert durchs Wohnzimmer wandern lassen? Kein großes Problem, denn der Commodore 128 verfügt, fast schon ebenso traditionell wie beim Datasetten-Anschluß, natürlich über den Userport.

Der Userport ist eine 8-Bit-Parallelschnittstelle, wobei der Benutzer frei definieren kann, welche Bits zur Ein- und welche zur Ausgabe benutzt werden sollen. Für Bastler und Hardware-Freaks ist diese Schnittstelle sehr willkommen, da flexibel. Hinzu kommt noch, daß man den Userport auch vom BASIC her mittels PEEK- und POKE-Kommandos programmieren kann. Zur Ablaufsteuerung verfügt man noch über zwei Handshake-Leitungen.

Die Belegung der Userport-Leitungen:

1	GND
2	+5V; mit maximal 100 mA belastbar
3	-RESET; mit der gleichnamigen Prozessorleitung verbunden
4	CNT1; verbunden mit CNT von CIA1
5	SP1; mit SP von CIA1 verbunden
6	CNT2; Leitung CNT von CIA2
7	SP2; verbunden mit SP von CIA2
8	-PC2; Handshake-Ausgang von CIA2
9	ATN OUT; Steuerleitung des seriellen IEC-Bus, stammt von PA3 der CIA2
10	9V AC; Wechselspannung; mit maximal 100 mA belastbar
11	Gegenpol zu 10
12	GND
A	GND
B	-FLAG2; Handshake-Eingang von CIA2
C-L	PB0-PB7; I/O-Lines von CIA2
M	PA2; I/O-Line von CIA2.
N	GND

Um Ihnen vorzuführen, wie Sie den Userport programmieren können, haben wir uns ein kleines Beispiel ausgedacht.

Unsere Beispielschaltung besteht lediglich aus vier Schaltern, vier Leuchtdioden, acht Widerständen und einem IC. Sie werden damit die Grundbegriffe der Datenein- und -ausgabe über den Userport sicherlich verstehen lernen. Die Schaltung finden Sie

am Ende dieses Abschnittes abgedruckt. Sie ist sehr einfach, daher haben wir sie hier nicht dokumentiert.

Wegen der Vielzahl der belegten Anschlüsse des Userports muß zunächst geklärt werden, welche Anschlüsse dem Benutzer nun überhaupt zur Verfügung stehen. Falls Sie nicht gerade eine RS232-Cartridge betreiben, können Sie die folgenden Anschlüsse ohne Rückwirkung auf die übrigen Funktionen des Rechners benutzen:

1-2, 4-8, 10-12, A-N

Auf unser Beispiel bezogen: Die Datenleitungen *PB0-PB7* lassen sich individuell auf Eingabe oder Ausgabe programmieren. Wir benutzen die Leitungen *PB0-PB3* als Eingabe und die Leitungen *PB4-PB7* als Ausgabe. Die Festlegung der Datenrichtung geschieht einfach durch das Laden des Datenrichtungsregisters für den Datenport B auf der Adresse 56579. Ein gesetztes Bit bedeutet Ausgabe auf dem korrespondierenden Bit des Datenports B (Adresse 56577), ein gelöschtes Bit bedeutet entsprechend Eingabe. Um für unser Beispiel die Datenrichtungen festzulegen, d.h. Bits 0-3 als Eingabe, 4-7 als Ausgabe, benutzen Sie folgendes Kommando:

```
POKE 56579,240
```

Damit sind die oberen Bits gesetzt und die korrespondierenden Bits des Datenports B stehen auf Ausgabe, die restlichen auf Eingabe.

Wie handhaben wir aber nun unsere kleine Schaltung weiterhin? Nichts leichter als das!

```
PRINT PEEK(56577) AND 15
```

signalisiert Ihnen den Zustand der vier Schalter, und mit dem Kommando

```
POKE 56577,X
```

können Sie die Leuchtdioden ein- und auch wieder ausschalten, wobei der Wert X sich nur aus den Werten 16, 32, 64 und 128

zusammensetzen darf - die unteren Bits sind ja nur zum Auslesen.

Sollten Sie bereits Eigenes planen - Sie wollen Ihre Familie entlasten, indem Sie die Waschmaschine an den Commdore 128 hängen u.ä. -, so beachten Sie unbedingt folgendes, wenn Sie Ihren Rechner dabei nicht beschädigen wollen:

Bei Verwendung des Userports als Eingang darf die Eingangsspannung nur im Bereich 0 bis 5 Volt liegen. Eine Spannung von 0 bis 0,6 Volt wird als Null interpretiert, eine Spannung von 1,6 bis 5 Volt als Eins. Alle Spannungen zwischen 0,7 und 1,5 Volt können zufällig als Null oder als Eins interpretiert werden.

Verwenden Sie den Userport als Ausgang, so beachten Sie bitte, daß die Ausgänge nur die Belastung eines TTL-Einganges aushalten. Sie könnten also keinesfalls eine Leuchtdiode direkt anschließen - dies würde nämlich langfristig zur Zerstörung der CIA führen. Es empfiehlt sich in jedem Fall immer eine Pufferstufe, wie auch in unserem Beispiel.

Vermeiden Sie unbedingt, ein auf Ausgabe programmiertes Portbit mit einer Fremdspannung von außen zu beaufschlagen, dies würde wohl zur unmittelbaren Zerstörung führen. Überlegen Sie sich deshalb gut, welche Werte Sie in das Datenrichtungsregister laden, damit Sie nicht versehentlich ein auf Eingabe vorgesehenes Bit auf Ausgabe programmieren.

Wenn Sie die Stromversorgung für Ihr Projekt entnehmen wollen, beachten Sie bitte, daß Sie die beiden zur Verfügung stehenden Spannungen nicht mit mehr als je 100 mA belasten. Bei leichten Übertretungen wird wohl zunächst der Kassettenrecorder streiken - danach verabschiedet sich die Sicherung im Inneren des Commodore 128 und evtl. auch die Primärsicherung im Trafogehäuse. Zerstört wird dabei (gottlob) jedoch nichts weiter.

Dies sollte nur eine kleine Anleitung zur Bedienung des Userports in einem einfachen Anwendungsfall sein. Wollen Sie für komplexere Aufgaben auch die anderen Leitungen benutzen, so orientieren Sie sich über ihre Handhabung bitte im Kapitel 3 (CIA).

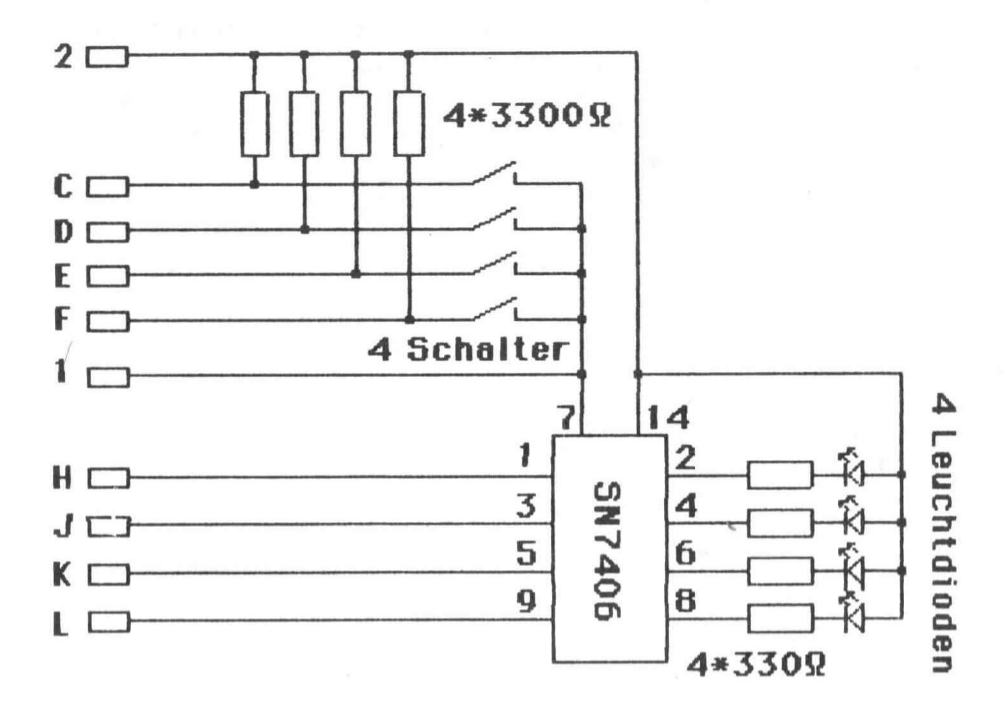

USER-Port
2
C
D
E
F
1
H
J
K
L
4*3300Ω
4 Schalter
7
14
1
2
3
4
5
6
9
8
SN7406
4*330Ω
4 Leuchtdioden

Abb. 2

1.4 Die RS232-Schnittstelle

Die RS232 - in Europa auch bekannt unter der Bezeichnung V24 - öffnet dem Commodore Tür und Tor. Die meisten Peripherie-Geräte haben einen RS232-Anschluß; beispielsweise der Laserdrucker, mit dem dieses Buch ausgedruckt worden ist.

Aber auch sonst ist die RS232 in aller Munde - wir befinden uns ja bekanntlich im Hacker-Zeitalter, und Telefon-Modems laufen eben genau über diese Schnittstelle.

RS232 ist die Bezeichnung für eine Schnittstelle zur seriellen Datenübertragung - parallele Datenübertragung per Telefon ist beispielsweise nicht möglich.

Bei der seriellen Übertragung werden die acht Bits eines Bytes nicht parallel, sondern Bit für Bit über die Leitung geschoben. Dies hat den Vorteil, daß man mit weniger Leitungen auskommt. Der Nachteil ist, daß dieses Verfahren langsamer ist - dafür wiederum eignet es sich zur Übertragung von Daten via Telefonleitung. Sie sehen, so hat alles seine zwei Seiten.

Das Betriebssystem des Commodore 128 enthält bereits die komplette Software zur Bedienung einer seriellen RS232-Schnittstelle - übrigens auch im 64er-Modus. Die Schnittstelle selbst ist als RS232-Steckmodul erhältlich, das man auf den Userport aufsteckt. Vom Modul wird die erforderliche Pegelumwandlung auf +/- 12 Volt selbsttätig übernommen.

Das Betriebssystem (im folgenden sind beide gleichermaßen gemeint) hat der RS232-Schnittstelle die Geräteadresse 2 zugeordnet. Wird ein logisches File mit dieser Geräteadresse eröffnet, so werden zwei Buffer mit der Länge von 256 Byte angelegt: ein Ein- und ein Ausgabebuffer. Im 128er-Modus sind diese Buffer an den Adressen $0C00 und $0D00 festgelegt. Im 64er-Modus zeigen zwei Pointer auf diese Buffer: $F7/F8 zeigt auf den RS232-Eingabebuffer und $F9/FA zeigt entsprechend auf den RS232-Ausgabebuffer. Ferner müssen Sie folgendes im 64er-Modus beachten:

Der angelegte Bufferbereich liegt meist im oberen Bereich des RAM. Wird in einem BASIC-Programm die RS232 verwendet,

sollte das Programm mit dem OPEN-Kommando beginnen, da dabei alle Variablen gelöscht werden. Ferner wird nicht geprüft, ob noch ausreichend Speicherplatz vorhanden ist. Beim CLOSE-Befehl wird dieser Buffer wieder freigegeben, allerdings werden hierbei wieder alle Variablen gelöscht, da ein CLR-Kommando ausgeführt wird (andere Dateien werden "vergessen"!) Darum sollte man erst am Ende des Programmes die Datei wieder schließen. Man kann immer nur eine RS232-Datei offen haben.

Beim Schließen eines RS232-Datenkanals wird eine eventuell laufende Übertragung abgebrochen und die Pufferanzeige zurückgesetzt. Wünschen Sie dies nicht, so können Sie durch das Kommando

```
SYS 61604 (JSR $F0A4)
SYS 59372 (JSR $E7EC)
```

(SYS 61604: *im 64er-Modus und*; SYS 59372: *im 128er-Modus*)

warten, bis der komplette Pufferinhalt übertragen worden ist. Darum sollten Sie dieses Kommando unbedingt vor dem CLOSE-Kommando ausführen.

Die Parameter für die Datenübertragung werden durch ein Kontrollregister und ein Befehlsregister festgelegt. Diese beiden Register werden beim Öffnen des Files mit dem Filenamen zusammen übergeben.

Das Kontrollregister dient der Definition der Baudrate und bestimmt die Anzahl der zu übertragenden Daten- und Stopp-Bits. Die Baudrate bestimmt die Geschwindigkeit der Datenübertragung. 1000 Baud bedeutet beispielsweise 1000 Bits in der Sekunde. Die Stoppbits werden nach jedem übertragenen Datenwort (5 - 8 Bits) gesendet.

Das Befehlsregister bestimmt die Übertragungsart, die Paritätsprüfung und die Art des Handshakes.

Beim Kontrollregister bestimmen die untersten 4 Bits die Baudrate nach folgender Tabelle:

Bit 3	2	1	0	dezimal	Baudrate
0	0	0	0	0	Anwender-Baudrate, s.u.
0	0	0	1	1	50
0	0	1	0	2	75
0	0	1	1	3	110
0	1	0	0	4	134.5
0	1	0	1	5	150
0	1	1	0	6	300
0	1	1	1	7	600
1	0	0	0	8	1200
1	0	0	1	9	1800
1	0	1	0	10	2400
1	0	1	1	11	3600 (n.i.)
1	1	0	0	12	4800 (n.i.)
1	1	0	1	13	7200 (n.i.)
1	1	1	0	14	9600 (n.i.)
1	1	1	1	15	19200 (n.i.)

(N.i.) bedeutet *nicht implementiert* - diese Geschwindigkeiten können von Ihrem Commdore nicht erreicht werden. Wir können also nur Baudraten zwischen 50 und 2400 programmieren. Die Anzahl der Datenbits wird durch Bit 5 und 6 bestimmt:

Bit 6	5	dezimal	Anzahl der Datenbits
0	0	0	8 Bits
0	1	32	7 Bits
1	0	64	6 Bits
1	1	96	5 Bits

Im ersten Byte bestimmt noch Bit 7 die Anzahl der Stoppbits:

Bit 7	dezimal	Anzahl der Stoppbits
0	0	1 Stoppbit
1	1	2 Stoppbits

Nachdem wir das erste Byte definiert haben, müssen wir noch das zweite Byte, das Befehlsregister, definieren.

Bit 0	dezimal	Handshake
0	0	3-Draht-Handshake
1	1	X-Draht-Handshake

Bit 4	dezimal	Übertragungsart
0	0	Vollduplex
1	16	Halbduplex

Bit 7 6 5	dezimal	Paritätsprüfung
X X 0	0	Keine Paritätsprüfung kein 8. Datenbit
0 0 1	32	ungerade Parität
0 1 1	96	gerade Parität
1 0 1	160	8. Datenbit immer 1 Keine Paritätsprüfung
1 1 1	224	8. Datenbit immer 0

Noch eine kleine Bemerkung zum Handshake: Haben Sie 3-Draht-Handshake gewählt, so werden die Steuerleitungen *CTS* (Clear To Send) und *DSR* (Data Set Ready) beim Senden und Empfangen nicht ausgewertet; dies aber genau bedeutet, daß der Rechner die Daten sendet, beispielsweise an einen Drucker, unabhängig davon, ob der Drucker die Daten schon verarbeiten kann oder nicht. Wollen Sie dies nicht, was bedeutet, das Gerät muß in der Lage sein, die Datenübertragung anzuhalten, so müssen Sie den X-Draht-Handshake auswählen. Dazu müssen die oben erwähnten Steuerleitungen verdrahtet werden; Voraussetzung ist, daß der Drucker überhaupt in der Lage ist, diese Leitungen bedienen zu können. Wenn man zwei Rechner miteinander verbindet, so reicht zumeist der 3-Draht-Handshake aus.

Lassen Sie uns zunächst einmal ein Beispiel durchgehen: Wir wollen einen RS232-Datenkanal mit folgenden Parametern eröffnen:

* *Übertragungsrate 2400 Baud*
* *7 Datenbits (ASCII)*
* *2 Stoppbits*
* *keine Paritätsprüfung*
* *8. Datenbit immer 0*
* *Vollduplex*
* *3-Draht-Handshake*

Nachdem Sie die Bits aus obigen Tabellen zusammengesucht haben, eröffnen Sie den Datenkanal mit folgender OPEN-Anweisung:

OPEN 1, 2, 0, CHR$(10+0+128) + CHR$(0+0+224)

Zumeist ist das zweite Byte in der OPEN-Anweisung allerdings CHR$(0).

1.4.1 Programmierung eigener Baudraten

Die verschiedenen Baudraten werden durch die Timer in den CIAs realisiert. Man kann aber auch Baudraten programmieren, die nicht in der Tabelle zu finden sind, beispielsweise 111 Baud. Allerdings kann man dabei nicht die angegebenen 2400 Baud überschreiten - hierzu ist die eingebaute Software zu langsam. Die CIAs (bzw. die Timer) lösen nach einer von der Baudrate abhängigen Zeit einen nicht maskierbaren Interrupt *(NMI)* aus, der die einzelnen zu empfangenen und zu sendenden Bits voneinander trennt. Wollen Sie nun eine eigene Baudrate verwenden, so können Sie die entsprechenden Timer-Werte als drittes und viertes Zeichen des Filenamens beim OPEN-Kommando übergeben. Die Timer-Werte lassen sich nach folgender kleiner Formel ermitteln:

T = 492662/BAUD - 101

Den Wert, den wir bei dieser Formel erhalten, müssen wir in Low- und in High-Byte trennen und dann als drittes und viertes Zeichen im Filenamen übergeben. In das Kontrollregister müssen wir anstelle der Baudrate dann eine Null definieren (Anwender-Baudrate), damit das Betriebssystem weiß, daß wir eine eigene Baudrate verwenden wollen.

Das nun folgende Beispiel verwendet dieselben Parameter wie oben, jedoch bei einer Baudrate von 1000.

```
100 BAUD=1000
110 T=492662/BAUD - 101
120 TH=INT(T/256): TL=T AND 255
130 OPEN 1,2,0,CHR$(128)+CHR$(224)+CHR$(TL)+CHR$(TH)
```

Man kann global sagen, daß man bei eigener Baudraten-Programmierung Baudraten von 8 bis 2400 erreichen kann.

1.4.2 Die Status-Abfrage

Will man bei der Übermittlung von Daten via RS232 feststellen, ob Fehler aufgetreten sind, so kann man sich, wie beim IEC-Bus, der Statusvariablen *ST* bedienen. Allerdings ist die Bedeutung von *ST* bei der RS232 etwas anders. Auch wird die Variable *ST* nach dem Auslesen durch BASIC gelöscht. Soll der Statuswert also für mehre Abfragen vorhanden sein, so ist ein Abspeichern in eine Zwischenvariable nötig.

Hier ist nun die bitweise Aufschlüsselung der Statusvariablen *ST*. Ein gesetztes Bit bedeutet, daß dieses Ereignis zutreffend ist.

Bit	*Beschreibung*
0	*Paritätsfehler*
1	*Rahmenfehler*
2	*Empfängerbuffer voll*
3	*Empfängerbuffer leer*
4	*CTS (Clear To Send) Signal fehlt*
5	*unbenutzt*
6	*DSR (Data Set Ready) Signal fehlt*
7	*Break-Signal empfangen*

Im 64er-Modus können Sie bei Programmierung in Maschinensprache angeben, wo die RS232-Ein- und Ausgabebuffer liegen sollen. Im 128er-Modus sind diese Buffer ja bereits recht praktisch angeordnet. Die Pointer für die Buffer liegen an den Adressen $F7-$FA (s.o.).

1.5 Der Cartridge-Port

Der Cartridge-Port - auch bekannt als Expansion-Bus - ist eine sinnvolle Einrichtung des Commodore. In diesen Port kann man ROM-Module einstecken, seien sie nun mit Spielen oder mit BASIC-Erweiterungen o.ä. bestückt. An dieser Schnittstelle stehen sowohl Adreß- als auch Datenbus des Rechners zur Verfügung (der Rechner ist an dieser Stelle deshalb auch sehr empfindlich).

Doch hier zunächst die Pinbelegung des 44poligen-Steckplatzes:

1	GND
2-3	+5V
4	-IRQ; mit der IRQ-Leitung des Prozessors verbunden
5	CR/-W; mit R/-W des Prozessors verbunden
6	DOT CLOCK; Punktrastertakt für den VIC; ca. 7,83 MHz
7	-I/O1; gewöhnlich =0 im Adreßsbereich $DE00 bis $DEFF
8	-GAME; Eingang zum AM (Address-Manager)
9	-EXROM; wie oben
10	-I/O2; gewöhnlich =0 um Bereich $DF00 bis $DFFF
11	-ROML; Ausgang vom AM
12	BA; Signal vom VIC, welches die Gültigkeit von Lesedaten anzeigt
13	-DMA; Eingang. 0=Bussystem für den externen Zugriff reservieren
14-21	CD7-CD0; Datenbus
22	GND
A	GND
B	-ROMH; Ausgang vom AM
C	-RESET
D	-NMI
E	02; Systemtakt Ausgang
F-Y	CA15-CA0; Adreßbus
Z	GND

Sowohl im 64er- als auch im 128er-Modus wird beim Einschalten des Rechners bzw. beim RESET getestet, ob der Cartridge-Slot belegt ist. Ist dies der Fall, so wird die Speicherkonfiguration im Adreß-Manager entsprechend darauf eingestellt, und die *Leitung* des Rechners übernimmt dann nicht das eingebaute ROM, sondern die Cartridge. Dies ist eine sehr bedienerfreundliche Lösung, da der Benutzer nur die Cartridge einschieben und den Rechner einschalten muß.

2. Der VIC-Chip

Wie Sie bereits wissen, verfügt der Commodore 128 über drei Buchsen zum Anschluß von Bildschirmen. Theoretisch können Sie drei Bildschirme gleichzeitig an Ihren Rechner anschließen, dies wäre aber unsinnig, weil Sie dann auf zwei Bildschirmen denselben Inhalt vorfänden.

Zwei dieser drei Anschlußbuchsen werden vom VIC-Chip versorgt. Der VIC-Chip hat sich schon millionenfach im Commodore 64 bewährt und ist wegen seiner vielen positiven Eigenschaften wie beispielsweise der Möglichkeit, Sprites sehr einfach zu programmieren und anzuzeigen, sehr beliebt. Der VIC-Chip im Commodore 128 hat allerdings zwei weitere Register erhalten, die wir später beschreiben wollen. Er übernimmt sowohl die Darstellung des 40-Zeichen-Zeichen-Modus als auch die Darstellung von Grafiken.

Am HF-Anschluß kann jedes Fernsehgerät über den Antenneneingang angeschlossen werden - der Computer spielt praktisch alternativer Fernsehsender. Dies ist eine relativ beliebte - da preiswerte - Lösung, muß man doch lediglich das entsprechende Kabel erwerben. Je nach Fernsehgerät ist die Bildqualität auch befriedigend, allerdings nicht für längeres Arbeiten am Gerät geeignet. Das liegt daran, daß die Trägerfrequenz erst vom Rechner aufmoduliert (es muß ja *gesendet* werden) und dann im Fernsehgerät wieder demoduliert werden muß (das Fernsehgerät verhält sich ja genauso, wie bei jedem anderen Sender auch - ob Dallas oder Commodore 128 ist egal). Unter dieser Mehrfachbehandlung des Bildes leidet die Auflösung dann entsprechend stark.

Hat sich das Portemonnaie vom Kauf des Commodore 128 wieder erholt, kommt eventuell ein Farbmonitor in Betracht, etwa der Commodore 1902. Vielleicht ist man ja auch Umsteiger, dann kann man natürlich sofort vom zweiten Anschluß fleißig Gebrauch machen: dem *Composite-Video-Ausgang*. Hier muß weder moduliert noch demoduliert werden, es ist die reine Bildinformation plus der Synchronimpulse verfügbar. Diese Monitore sind zwar ein wenig teurer, bieten dafür

aber eine erheblich bessere Bildqualität, da die Bildauflösung um einiges besser ist. Nachteil: Beim abendlichen Streit, ob Fußball oder Hobbythek gesehen wird, kann man nicht auf den Monitor als Fernsehempfängeralternative zurückgreifen.

Der VIC-Chip liegt im Commodore 128 an derselben Adresse wie im 64er - logischerweise, schließlich wird er im 64er-Modus auch angesprochen und muß der Kompatibilität wegen dieselben Adressen aufweisen.

Startadresse: $D000

Allerdings muß der VIC-II-Chip (wir wollen ihn VIC-II nennen, da er ja nicht 100%-ig identisch mit seinem Vorgänger ist) bei 2 MHz Taktfrequenz (Fast-Modus) die Waffen strecken. Der VIC-II-Chip ist im Commodore 128 - wie auch in seinem Vorgänger - für die Taktversorgung zuständig. Wie Sie vielleicht wissen, nutzten die VIC-Chips die Taktlücken (Zeiten, in denen der Prozessor nicht auf den Speicher zugreift) aus, um sich zur Auffrischung des Videobildes ein Zeichen aus dem Video-RAM zu holen. Dies wird gemacht, um die Arbeitsgeschwindigkeit des Prozessors nicht zu beschneiden. Wenn der Prozessor nun mit 2 MHz getaktet wird, wird sowohl die Arbeitsgeschwindigkeit verdoppelt als auch die Taktlücke (logischerweise) halbiert. Es reicht nun die Taktlücke für den VIC nicht mehr aus, um auf den Speicher zuzugreifen. Der VIC-II-Chip schaltet die Videoausgabe ab; Sie erhalten ein einfarbiges Bild (wie es einige von Ihnen vom Laden von Kassette kennen). Der Videocontroller - für die Darstellung des 80-Zeichen-Bildschirmes zuständig - ist davon nicht betroffen. Er stellt weiterhin fleißig seine 80 Zeichen pro Zeile dar. Das Umschalten von 1 auf 2 MHz funktioniert übrigens auch im 64er-Modus!! Dazu muß man Bit 0 des Registers 48 setzen.

```
POKE 53296,1 entspricht dem Kommando FAST
POKE 53296,0 entspricht dem Kommando SLOW
```

Diese zwei POKEs lassen sich also auch vom 64er-Modus aus anwenden. Allerdings unterscheidet sich das Kommando FAST doch ein wenig von o.g. POKE-Kommando: Beim BASIC-Befehl FAST wird zusätzlich automatisch der 40-Zeichen-Bildschirm

abgeschaltet, so daß man das durch den 2-MHz-Modus entstehende bunte Wirrwarr auf dem Bildschirm nicht mitansehen muß.

Der VIC-Chip erfüllt nicht nur alle zur Erzeugung eines Bildes notwendigen Aufgaben, sondern kümmert sich zusätzlich um das gesamte Timing für den dynamischen Speicher, was den Prozessor nicht unwesentlich entlastet.

Doch hier erst einmal einige Features zum VIC-Chip:

* *16 Farben*
* *40 Zeichen und 25 Zeilen*
* *Grafikfähig mit 320 x 200 Punkten (Hi-Res-Modus)*
* *Vierfarbengrafik mit 160 x 200 Punkten (Multi-Color-Modus)*
* *Multi-Color-Modus im Textmodus möglich*
* *Darstellung und Verwaltung von 8 Sprites*
* *Raster- und Spritekollisionsinterrupt*
* *Erzeugung eines normgerechten PAL-Signals*
* *Verschiebbarer Video-RAM und Zeichengenerator*
* *Selbständiges Handling von 16 KByte dynamischen RAM*

Die Pinbelegung des VIC-Chip:

1-7	D6-D0; Prozessordatenbus.
8	-IRQ; 0 wenn ein Bit des IMR und des IRR übereinstimmen.
9	-LP; Eingang, Light-Pen-Strobe.
10	-CS; Prozessorbusaktionen finden nur während CS=0 statt.
11	R/-W; 0=Übernahme der Daten vom Bus.
12	BA; =0 wenn Daten bei einem Lesezugriff noch nicht bereitstehen.
13	VDD; +12V
14	COLOR; Farbinformation Ausgang
15	SYNC; Zeilen- und Bildsynchronisationsimpulse
16	AEC; 0= VIC benutzt Systembus, 1= Bus frei
17	0OUT; Ausgang Systemtakt
18	-RAS; dyn. RAM Steuersignal
19	-CAS; wie oben
20	GND
21	0COLOR; Eingang Farbfrequenz
22	0IN; Eingang Dotfrequenz
23	A11; Prozessoradressbus
24-29	A0/A8-A5/A13; gemultiplexter (Video-) RAM-Adreßbus

30-31 A6-A7; (Video-) RAM-Adreßbus
32-34 A8-A10; Prozessoradreßbus
35-38 D11-D8; Daten aus Farbram
39 D7; Prozessordatenbus
40 VCC; +5V
41-44 K0-K3; Keyboard-Interface-Leitungen. Diese Leitungen gehen direkt an die erweiterte Tastatur.

2.1 Registerbelegung des VIC-Chip

Der VIC-II-Chip verfügt über 49 Register, die an der Adresse $D000 + Registernummer liegen und angesprochen werden können. Diese Register sind hier im einzelnen beschrieben:

REG 0 Sprite-Register 0: X-Koordinate
Hier sind 8 Bits der X-Bildschirmkoordinate des Sprite 0 enthalten. Bit 9 befindet sich im Register 16 des VIC-Chip.

REG 1 Sprite-Register 0: Y-Koordinate
In diesem Register befindet sich die Y-Position des Sprite 0. Allerdings braucht die Y-Koordinate keinen Übertrag (9. Bit), da die Y-Koordinate maximal den Wert 199 annehmen kann.

Es folgen an dieser Stelle die Register 2 bis 15 entsprechend für die Sprites Nr. 1 bis 7. Jedes Sprite hat ein Registerpaar im VIC-Chip: Sprite 0 hat das Registerpaar 0/1, Sprite 1 das Paar 2/3 ... Sprite 7 das Paar 14/15.

REG 16 MSB der X-Koordinaten
In diesem Register werden die Überträge der X-Koordinaten der Sprites abgelegt. Gesetztes Bit bedeutet MSB (9. Bit) des entsprechenden Sprites gesetzt, 0 bedeutet nicht gesetzt. Das MSB von Sprite 0 ist durch Bit 0 repräsentiert, das MSB von Sprite 7 entsprechend durch Bit 7.

REG 17 Steuerregister 1
Bit 0-2 : Offset des oberen Bildrandes in Rasterzeilen,
Bit 3 : 0= 24 Zeilen, 1= 25 Zeilen,
Bit 4 : 0= Bildschirm aus,
Bit 5 : 1= Standard Bit-Map-Mode (Grafik),
Bit 6 : 1= Extended Color Mode (Text),
Bit 7 : Übertrag aus REG 18.

REG 18 Raster-IRQ
Nummer der Rasterzeile, bei deren Strahldurchlauf ein Raster-IRQ ausgelöst werden soll. Das 8. Bit der Rasterzeile befindet sich im REG 17.

REG 19 X-Anteil der Bildschirmposition, an der sich der Strahl gerade befand, als ein Strobe ausgelöst wurde.

REG 20 Wie REG 19, jedoch Y-Anteil.

REG 21 Sprite Enable
In diesem Register wird angegeben, ob ein Sprite eingeschaltet ist (Bit 1) oder nicht (Bit 0). Sprite 0 wird durch Bitposition 0 repräsentiert, Sprite 7 durch Bit 7 des Registers.

REG 22 Steuerregister 2
Bit 0-2 : Offset vom linken Bildrand in Rasterpunkten,
Bit 3 : 0=38 Zeichen, 1=40 Zeichen (horizontal),
Bit 4 : Multi-Color-Modus (Grafik).

REG 23 Sprite Expand X
Die Sprites können in X-Richtung durch Setzen des entsprechenden Bits verdoppelt werden.

REG 24 Basisadresse von Zeichengenerator und Video-RAM
Bit 1-3 : Adreßbits 11-13 für Zeichenbasis,
Bit 4-7 : Adreßbits 10-13 für Video-RAM.

REG 25 IRR: Interrupt Request Register
In diesem Register wird angegeben, welche Register als Interrupt-Auslöser in Frage kommen sollen.
Bit 0 : Auslöser ist REG 18,
Bit 1 : Auslöser ist REG 31,
Bit 2 : Auslöser ist REG 30.
Bit 3 : Auslöser ist Pin LP
Bit 7 : =1 wenn mindestens ein anderes Bit 1 ist.

REG 26 IMR: Interrupt Mask Register
Belegung wie REG 25. Bei Übereinstimmung mindestens eines Bits aus IRR und IMR (IRR AND IMR<>0) wird ein Interrupt ausgelöst (Pin IRQ=0).

REG 27 Prioritätsregister (Sprites)
Ist das entsprechende Bit gesetzt, so hat das Hintergrundzeichen Vorrang vor dem Sprite.

REG 28 Multi-Color-Register (Sprites)
Ist das für ein Sprite repräsentative Bit gesetzt, so ist dieses Sprite in Multicolor dargestellt.

REG 29 Sprite-Expand Y
Die Sprites können durch Setzen des entsprechenden Bits in Y-Richtung doppelt so groß dargestellt werden.

REG 30 Sprite/Sprite-Kollision
Jedem Sprite ist ein Bit zugeordnet. Berühren sich zwei Sprites,so werden die beiden entsprechenden Bits gesetzt. Diese Bits bleiben gesetzt, bis man sie ausdrücklich löscht! Gleichzeitig wird im IRR Bit 2 gesetzt. Ist auch im IMR Bit 2 gesetzt, so wird ein Interrupt ausgelöst.

REG 31 Sprite/Background-Kollision
Jedem Sprite ist ein Bit zugeordnet. Berührt ein Sprite den Hintergrund (Background), so wird das entsprechende Bit gesetzt. Die Bits bleiben gesetzt, bis sie ausdrücklich rückgesetzt werden! Im IRR wird Bit 3 gesetzt; ist auch Bit 3 im IMR gesetzt, so wird ein Interrupt ausgelöst.

REG 32 Exterior Color (Rahmenfarbe)
In diesem Register wird die Rahmenfarbe gesetzt (0-15).

REG 33 bis REG 36 Background-Color-Register 0-3 (Hintergrundfarben)
Hintergrundfarbregister 0 bestimmt die Hintergrundfarbe im "normalen" Textmodus. Ist der Multi-Color-Modus eingeschaltet, so wird ebenfalls auf die Register 1-3 zugegriffen.

REG 37 und REG 38 Sprite Multi-Color-Farbe 0/1
Sprites, die in Multi-Color dargestellt werden, können die Hintergrundfarbe annehmen, die Spritefarbe oder die Multi-Color-Farbe 0 und 1.

REG 39 bis REG 46 Color Sprite 0-7
In diesen Registern werden die Farben der einzelnen Sprites abgelegt.

REG 47 Keyboard-Control-Register
Dieses Register beinhaltet den Status der vier Keyboard-Interface-Pins K0 bis K3. Bits 0 bis drei sind dafür zuständig. Bits 4-7 sind unbenutzt und werden beim Lesen auf 1 sein.

REG 48 2-MHz-Bit
Bit 0 dieses Registers gibt an, ob mit einem oder mit zwei Megahertz gefahren wird. Bits 1-7 sind unbenutzt. Bei gesetztem Bit werden alle Zugriffe vom VIC-II-Chip auf den Speicher abgebrochen, ausgenommen ist der Refresh des dynamischen RAMs.

Folgendes muß unbedingt vorweggenommen werden: Alle nun folgenden Beispielprogramme geben Sie bitte im 64er-Modus ein, den Sie durch *GO 64* erreichen. Dies ist nötig, weil der BASIC-Interpreter Eingaben praktisch *unwirksam* macht. Schalten Sie beispielsweise die Grafik durch die nötigen POKE-Anweisungen ein, so werden Sie lediglich ein Aufblitzen auf Ihrem Bildschirm sehen. Dasselbe gilt für die Programmierung der Sprites etc. Das liegt daran, daß der Interpreter durch Interrupts gesteuert seinen Willen durchsetzen will (und muß). Beispielsweise können Sie durch das MOVSPR-Kommando eine gleitende Bewegung eines Sprites erzeugen; dies geht nur mit der Interrupt-Technik. Wir sagen Ihnen im Tips-und-Tricks-Teil aber noch, wie man diese Steuerung umgehen kann.

Aber selbst die Koordinaten der Sprites, die Farbregister und vieles mehr wird vom Interpreter stets korrigiert. Sehr hinderlich ist dies, will man die Register des VIC direkt programmieren und nicht die "Kopien" des BASIC.

Wie bereits erwähnt, geben Sie die Beispielprogramme entweder im 64er-Modus ein, oder Sie schreiben als erste Zeile in alle folgenden Programme das folgende Kommando:

```
0 POKE 216, 255
```

Mit dieser POKE-Anweisung wird der BASIC-Interrupt aufgefordert, seine Korrekturen zu unterlassen.

Alle folgenden Beispielprogramme bedienen sich keiner speziellen BASIC-7.0-Kommandos, damit die Programme auch im 64er-Modus benutzt werden können, und um eine möglichst große Maschinennähe zu erlangen. POKE-Anweisungen eignen sich hier besonders, da sie leicht in Maschinensprache zu übersetzen sind.

2.2 Die Betriebsarten des VIC

Wie Sie vielleicht schon wissen und wie man den Registern entnehmen kann, gibt es einige Möglichkeiten der Bildschirmgestaltung mit dem VIC. Im 128er-Modus haben Sie dank der komfortablen BASIC-Kommandos nicht soviel Mühe, diese

Möglichkeiten auszuschöpfen, wie im 64er-Modus. Im letzteren müssen Sie mittels langer und komplizierter POKE-Befehle die verschiedenen Modi umschalten, was nebenbei auch nicht gerade der Übersichtlichkeit der Programme dienlich ist. Auch die Programmierung der Sprites ist im 64er-Modus bei weitem nicht so komfortabel wie im 128er-Modus, in dem man ja Sprites fließend durch das Kommando MOVSPR bewegen kann. Sollten Sie sich sagen: "Dann interessiert mich der nähere Aufbau des VIC-Chips nicht, denn ich will ohnehin nicht im 64er-Modus programmieren", so haben Sie nur bedingt recht. Wollen Sie nämlich in Maschinensprache programmieren, so kommen Sie nicht drum herum, sich mit der Registerbelegung etwas näher zu befassen, was wir nun tun wollen:

2.3 Sprites

Sprites sind bewegliche, frei definierbare Figuren mit einer Auflösung von 24 mal 21 Punkten. Sie können Sprites entweder zweifarbig (Spritefarbe und Hintergrundfarbe) oder im Multi-Color-Modus (4 Farben, dafür aber nur halbe Auflösung mit 12 mal 21 Punkten) darstellen. Im VIC-Chip sind 8 Sprites vorgesehen, die gleichzeitig über den Bildschirm bewegt werden können. Die Sprites können ihre Positionen im Rahmen von 512 mal 256 Rasterpunkten einnehmen, was bedeutet, daß man Sprites durchaus auch außerhalb des Bildschirmes bewegen kann.

Ist ein Sprite im Zweifarbenmodus definiert, so bedeutet gesetztes Bit einen gesetzten Punkt in der für dieses Sprite definierten Farbe. Ein nicht gesetztes Bit bedeutet transparent, also Hintergrundfarbe wird dargestellt. Im Multi-Color-Modus werden je zwei Bits zu einem Punkt zusammengefaßt, was bedeutet, daß man 4 Farben definieren kann. Die möglichen Bitkombinationen bedeuten im einzelnen folgende Farbe:

00: Transparent, Hintergrundfarbe (REG 33)
01: Multi-Color-Register 0 (REG 37)
11: Multi-Color-Register 1 (REG 38)
10: Sprite-Farbregister (REG 39-46)

Sie sehen, daß zwei Farben (Multi-Color-Register 0 und 1) jeweils für alle Sprites gleichzeitig definiert werden. Die Sprites

können sich höchstens in einer Farbe unterscheiden. Lassen Sie uns aber einmal ein Sprite *zu Fuß* definieren, d.h. wir bedienen uns nicht der bequemeren BASIC-Kommandos, sondern programmieren alles in BASIC, das sowohl im 128er- als auch im 64er-Modus ausgeführt werden kann. Zuerst müssen wir ein Sprite mittels DATA-Zeilen definieren (der Sprite-Editor existiert jetzt für uns nicht). Diese DATA-Zeilen sollen wie folgt aussehen:

```
1000 DATA 000,000,000
1010 DATA 000,000,000
1020 DATA 000,000,000
1030 DATA 000,000,000
1040 DATA 000,000,000
1050 DATA 000,000,000
1060 DATA 000,000,000
1070 DATA 003,255,255
1080 DATA 000,002,000
1090 DATA 192,170,128
1100 DATA 194,150,080
1110 DATA 234,150,080
1120 DATA 194,170,168
1130 DATA 192,170,168
1140 DATA 000,032,128
1150 DATA 000,170,160
1160 DATA 000,000,000
1170 DATA 000,000,000
1180 DATA 000,000,000
1190 DATA 000,000,000
1200 DATA 000,000,000
```

Im Normalfall malt man sich vor der Programmierung eines Sprites die Figur, die man darzustellen wünscht, auf einem Blatt Papier auf und unterteilt diese dann in die genannten 24 mal 21 Punkte. Es ergeben sich 21 Zeilen zu je 24 Punkten. Diese 24 Punkte teilt man in drei 8er-Päckchen ab, die man bekanntlich als ein Byte abspeichern kann. Jedes ausgefüllte Kästchen bedeutet gesetztes Bit, ein leeres Kästchen bedeutet ungesetztes Bit. Bei den Multi-Color-Sprites sieht es etwas schwieriger aus: Da müssen Sie anhand einer selbstdefinierten Farbentabelle eine der vier Bitkombinationen einsetzen. *Achtung: Sie müssen sich vorher genau überlegen, welche Farbe Sie als gemeinsame für*

alle Sprites definieren und welche Sie als individuelle Farbe für das einzelne Sprite haben wollen. Nach getaner Arbeit können Sie die einzelnen Bytes ausrechnen und aufschreiben. Diese notierten Werte ergeben dann eine DATA-Zeilen-Kolonne, wie in unserem Beispiel. Unser Beispiel-Sprite stellt einen Hubschrauber dar, so viel vorweg. Sie können es auf dem Papier natürlich nicht erkennen (oder Sie sind schon so sehr Insider, daß es bereits gefährliche Ausmaße angenommen hat). Natürlich ist die Erstellung von Sprites mit Spritegeneartoren einfacher; ein solcher ist om 128er-Modus enthalten.

2.3.1 Adresse des Sprites

Nun gut: Wir haben unsere Daten und müssen diese irgendwo im Speicher ablegen. Es gibt für jedes Sprite einen Pointer, der dem VIC-Chip mitteilt, wo er das jeweilige Sprite vorfindet. Diese Pointer befinden sich an den Adressen 2040 bis 2047, schließen also unmittelbar an das Video-RAM an. Pro Sprite benötigen wir 3 x 21 = 63 Bytes. Sie haben schon richtig erkannt, daß der Pointer lediglich aus einem Byte besteht und somit also keine absolute Adresse angeben kann. Es wird vielmehr die Position in *Pointer mal 64* angegeben, was für exakt 16 KByte ausreicht, mehr kann der VIC-Chip ohnehin nicht auf einmal adressieren. Sollten Sie die Startadresse des Video-RAM verschieben, so verschieben sich natürlich auch die Pointer für die Sprites sowie deren Startadressen im entsprechenden Ausmaß! Geben wir beispielsweise einmal an, daß Sprite Nummer 1 an Adresse 13 * 64 = 832 definiert ist.

```
POKE 2041,13
```

Adresse:	2040	2041	2042	2043	2044	2045	2046	2047
Sprite#:	0	1	2	3	3	4	5	6

Sie könnten natürlich durchaus auch weitere Sprites auf diese Speicheradresse legen, was bedeuten würde, daß mehrere Sprites dasselbe Aussehen hätten. Doch um unser Beispiel-Sprite darzustellen, müssen wir zuerst die vorhandenen Werte auch in die Speicheradressen POKEn:

```
10 FOR I=0 to 62
20  READ D
30  POKE 13*64+I,D
40 NEXT
50 POKE 2041,13: REM Sprite 1 an Adresse 832
```

2.3.2 Einschalten

Wenn Sie das Programm starten, so werden Sie feststellen, daß unser Programm noch nicht ausreicht. Wir müssen nämlich das Sprite noch explizit einschalten! Dies machen wir am besten durch logisches ODER-Verknüpfen des entsprechenden Bits im Register 21, da durch direktes POKEn andere Sprites evtl. gelöscht würden.

POKE 53248+21,PEEK(53248+21) OR 2

schaltet Sprite 1 ein. Wollen Sie beispielsweise Sprite 0 und 7 einschalten: *POKE 53248+21,PEEK(53248+21) OR 1 OR 128*, oder besser gleich: *POKE 53248+21,PEEK(53248+21) OR 129*. Sollten Sie wiederum Sprite 1 ausschalten wollen:

POKE 53248+21,PEEK(53248+21) AND NOT(2)

Wenn Sie mehrere Sprites gleichzeitig ausschalten wollen, beispielsweise Sprite 0 und 7:

POKE 53248+21,PEEK(53248+21) AND NOT(1 OR 128)

so erreicht man dies durch logisches ODER-verknüpfen der zu löschenden Sprites, die man anschließend negiert und dann AND-verknüpft. Wir schalten unser Sprite im Beispielprogramm ein:

60 POKE 53248+21, 2: REM Sprite 1 einschalten

```
Sprite: 7 6 5 4 3 2 1 0
Bit:    7 6 5 4 3 2 1 0
```

2.3.3 Farbe

Wir wollen aber auch noch die Farbe des Sprites definieren, da man sonst unter Umständen überhaupt nichts sieht:

```
70 POKE 53248+39+1, 5: REM Grüne Farbe
```

Dies geschieht in den Registern 39 bis 46: Register 39 definiert die Farbe für Sprite Nr. 0; Register 46 definiert entsprechend die Farbe für Sprite Nr. 7.

Folgende Farben stehen Ihnen zur Verfügung:

0	*Schwarz*	*8*	*Orange*
1	*Weiß*	*9*	*Braun*
2	*Rot*	*10*	*Hellrot*
3	*Türkis*	*11*	*Grau 1*
4	*Violett*	*12*	*Grau 2*
5	*Grün*	*13*	*Hellgrün*
6	*Blau*	*14*	*Hellblau*
7	*Gelb*	*15*	*Grau 3*

2.3.4 Position

Nachdem Sie diese Angaben gemacht und das Programm mit *RUN* gestartet haben, werden Sie immer noch nichts sehen, da die Position des Sprites sich außerhalb des Bildschirmes befindet. Um die X- und Y-Position des Sprites 1 anzugeben, müssen wir die Register 2 und 3 beschreiben:

```
80 POKE 53248+2, 50: REM X-Koordinate
90 POKE 53248+3, 70: REM Y-Koordinate
```

Sie können Ihr Sprite durch Schleifensteuerung etc. nun über den gesamten Bildschirm bewegen. Viele Leser stöhnen nun vielleicht schon auf: Sie erkennen, was BASIC 7.0 Ihnen alles an Arbeit bei der Programmierung von Sprites abnimmt. Es kommt aber noch "dicker": Wollen Sie beispielsweise Ihr Sprite an der X-Koordinate 310 positionieren, so reichen 8 Bits leider nicht mehr aus. Hierzu müssen Sie das neunte Bit des entsprechenden Sprites im Register 16 setzen (bzw. rücksetzen, wenn das Sprite

sich von rechts nach links bewegen soll). Wir positionieren unser Sprite an X-Koordinate 310:

POKE 53248+16, 2: REM Sprite 1 - 8. Bit setzen

Wollen Sie vermeiden, daß andere Sprites von diesem Kommando in Mitleidenschaft gezogen werden, so müssen Sie wieder explizit das entsprechende Bit ansprechen:

POKE 53248+16, PEEK(53248+16) OR 2

Lassen Sie unser Sprite einmal von links nach rechts über den Bildschirm wandern:

```
FOR I=0 TO 400
 POKE 53248+2, I AND 255:REM Unteren 8 Bit ausmaskieren
 POKE 53248+16, PEEK(53248+16) AND NOT 2 OR 2*ABS(I>255)
NEXT I
```

Etwas komplizierter ist die vorletzte Zeile: Es wird das MSB des Sprites 1 durch AND NOT 2 auf alle Fälle erst einmal rückgesetzt. Durch das Anhängsel OR 2*ABS(I>255) wird das entsprechende Bit allerdings wieder gesetzt, wenn es nötig sein sollte (X-Koordinate ist größer 255). Dies alles geschieht sogar, ohne die anderen Bits zu beeinflussen.

2.3.5 Vergrößern

Eine weitere wichtige und teilweise recht nützliche Möglichkeit ist, die Sprites in horizontaler und/oder vertikaler Richtung in der Größe zu verdoppeln. Der VIC-Chip stellt für diesen Zweck zwei Register zur Verfügung: *X-Expand* und *Y-Expand*. Auch hier wird wieder jedes Sprite durch ein Bit repräsentiert. Durch Setzen dieses Bits wird das entsprechende Sprite in X- oder Y-Richtung vergrößert. Wir wollen unser Sprite nun sowohl in X- als auch in Y-Richtung verdoppeln:

POKE 53248+23,2: REM Verdopple Sprite 1 in Y-Richtung
POKE 53248+29,2: REM Verdopple Sprite 1 in X-Richtung

Da wir unsere Sprites sowohl in X- als auch in Y-Richtung verdoppeln können, besteht also die Möglichkeit, die Sprites um den Faktor Vier zu vergrößern. Sicherlich eine interessante Sache bei der Realisierung einer Laufschrift.

2.3.6 Hintergrund

Sicherlich haben Sie beim Eingeben des einen oder anderen Beispiels bemerkt, daß beim Scrollen des Bildschirmes die Sprites nicht mitgescrollt wurden. Auch bleiben die Sprites beim Löschen des Bildschirmes weiterhin sichtbar. Die Sprites sind also ausschließlich durch ihre Position bestimmt. Wollen Sie ein Sprite aus dem Bildschirm haben, so können Sie es

a) ausschalten und
b) außerhalb des Bildschirmes positionieren.

Sprites haben noch eine weitere bemerkenswerte Eigenschaft: Wenn Sie mit dem Textcursor über ein Sprite fahren und Text darüber schreiben, so verdeckt das Sprite die Buchstaben - die Buchstaben sind nur an den Stellen sichtbar, an denen das Sprite transparent ist. Es entsteht fast der Eindruck eines 3-dimensionalen Bildes, da man sich Sprites und Hintergrund (so wollen wir die Nicht-Sprites einmal nennen) als zwei verschiedene Schichten vorstellen kann, die auch vollkommen anders behandelt werden. Es ist möglich, dem VIC-Chip mitzuteilen, daß wir einzelne Sprites nun aber nicht im Vordergrund haben wollen. Es gibt für jedes Sprite eine Prioritätsstufe, die dem VIC mitteilt, ob das Sprite oder der Hintergrund Vorrang hat. In unserem Beispiel würden die Buchstaben sich durchsetzen und das Sprite überdecken. Um ein Sprite hinter den Hintergrund zu bewegen, muß das entsprechende Bit im Register 27 gesetzt werden. Wir wollen unserem Hubschrauber einmal keine Priorität geben:

```
POKE 53248+27,2
```

Und schon hat sich der Hubschrauber hinter den Buchstaben versteckt. Um ihn wieder hervorzuholen, müssen wir lediglich das Bit wieder rücksetzen:

POKE 53248+27,0

Register 27: Hintergrundpriorität

```
Bit:    7 6 5 4 3 2 1 0
Sprite: 7 6 5 4 3 2 1 0
```

Sie haben sicherlich schon festgestellt, daß alle Register auf dieselbe Art und Weise aufgebaut sind: Ein Byte reicht aus, um alle 8 möglichen Sprites zu repräsentieren. Dabei steht Bit 0, das niederwertigste Bit, immer für Sprite 0 (Bit 7 entsprechend für Sprite 7).

Sie fragen sich, was wohl passiert, wenn mehrere Sprites übereinander liegen? Auch hierfür gibt es feste Regeln, allerdings können wir hierauf keinerlei Einfluß nehmen. Sollten sich zwei oder mehr Sprites überlagern, so liegt das Sprite mit der niedrigsten Nummer *oben*. Sollten sich also zum Beispiel Sprite 0 und Sprite 6 überlagern, so würde man von Sprite 0 alles sehen, während man von Sprite 6 bestenfalls die Silhouette sieht. Sprite 6 liegt also auf Sprite 7, Sprite 5 auf Sprite 6 ... bis dann Sprite 0 auf Sprite 1 thront. Je niedriger also die Spritenummer, desto höher die Priorität!

2.3.7 Kollision: Sprite-Sprite

Es ist auch möglich, daß sich zwei Sprites überlagern, d.h. daß sie wenigstens einen gemeinsamen Punkt besitzen. Oft ist es von Vorteil, wenn man von einer solchen Berührung erfahren kann, beispielsweise bei Spielen. Der VIC-Chip hat auch hier ein Register parat: Das Register 30 gibt uns Auskunft, ob und welche Sprites kollidiert sind. Wenn beispielsweise Sprite 0 und Sprite 6 kollidierten, so werden Bit 6 und Bit 0 im Register 30 gesetzt. Treffen mehr als zwei Sprites aufeinander, so werden entsprechend mehr Bits gesetzt. In unserem Beispiel - Sprite 0 und Sprite 6 treffen aufeinander - würden wir folgendes Ergebnis erhalten:

PRINT PEEK(53248+30)
65

Die Zahl 65 setzt sich zusammen aus gesetztem Bit 0 und gesetztem Bit 6: 64 + 1 = 65. Nachdem Sie das Register 30 ausgelesen haben, sollten Sie es unbedingt wieder auf Null zurücksetzen, da dies nicht automatisch geschieht. Die Folge wäre, daß Sie eine weitere Kollision nicht von der erkannten unterscheiden könnten - also folgt unserem o.g. Beispiel:

```
POKE 53248+30,0
```

2.3.8 Kollision: Sprite-Hintergrund

Sprites können natürlich auch Hintergrundzeichen berühren. So ist es dann auch möglich abzufragen, ob unser Hubschrauber den Cursor berührt oder nicht. Diese Abfrage ist davon unabhängig, ob jetzt das Sprite oder das Hintergrundzeichen Vorrang hat. Sollte ein Sprite ein Hintergrundzeichen berühren, so wird das entsprechende Bit im Register 31 gesetzt. Es gilt hier sonst dasselbe wie für Register 30: Löschen des Registers nach dem Auslesen ist oberstes Gebot. Sie wissen allerdings lediglich, daß ein Sprite ein Hintergrundzeichen berührt, Sie wissen nicht, welches Hintergrundzeichen betroffen ist, doch das ist zumeist egal. Allerdings besteht die Möglichkeit, dies anhand der Position zu bestimmen.

2.3.9 Multi-Color-Sprites

Sicherlich ein "Sahnehäubchen" auf die Möglichkeiten der Spriteprogrammierung ist die Möglichkeit der Definition von Sprites in Multi-Color. *Multi-Color* bedeutet nichts anderes als vierfarbig. Eine dieser Farbe ist die Hintergrundfarbe, zwei weitere Farben sind für alle acht Sprites identisch. Nur eine Farbe ist für jedes Sprite frei definierbar. Wollen Sie mehrere Sprites in Multi-Color darstellen, so ist vorher sorgsam zu überlegen, welche Farben in allen Sprites vorkommen sollen. Diese müssen Sie dann als die zwei fixen Sprite-Farbregister definieren. Ein kleines Manko haben Multi-Color-Sprites allerdings: Die Auflösung halbiert sich. Doch ist dies kaum als Manko zu bezeichnen, da die Auflösung in aller Regel mehr als ausreicht. Sie haben nun also eine Auflösung von 12 mal 21 Punkten. Die Größe des Sprites bleibt allerdings erhalten, da die

Punkte doppelt so breit werden - zwei Bits definieren je eine Farbe. Die verschiedenen Bitkombinationen haben im einzelnen folgende Bedeutung:

00	*Der Punkt hat die Hintergrundfarbe. (Man sieht also keinen Punkt).*
01	*Die Farbe wird aus Register 37 geholt.*
10	*Die Farbe wird aus dem jeweiligen Sprite-Farbregister geholt.*
11	*Die Farbe wird aus Register 38 geholt.*

Wir müssen dem VIC-Chip allerdings auch mitteilen, welche Sprites Multi-Color-Sprites sind. Dies geschieht, natürlich wieder bitweise, in Register 28. Wir wollen unseren Hubschrauber einmal zum Multi-Color-Sprite befördern:

```
POKE 53248 + 28, 2
```

Und siehe da: Er leuchtet in schillernden Farben. (Oder nicht?) Jetzt können wir es ja verraten: Der Hubschrauber sah nur deshalb so häßlich aus, weil wir ein Multi-Color-Sprite definiert haben. Die verschiedenen Bitkombinationen haben natürlich beim einfarbigen Sprite einen anderen Charakter als beim MC-Sprite. Als kleinen Leckerbissen wollen wir Ihnen noch ein Beispielprogramm anbieten, das unseren Hubschrauber zum Leben erweckt. Sie können auch an diesem Programm erkennen, wie man Sprites programmieren sollte, sei es nun in BASIC oder entsprechend in Maschinensprache.

```
10 REM SPRITE DEMONSTRATIONSPROGRAMM - DER HUBSCHRABSCHRAB
20 V=53248: STARTADRESSE DES VIC-CHIP
30 POKE V+32,15: POKE V+33,14: REM HINTERGRUNDFARBEN
40 PRINT "<Ctrl-7>"            : REM BETAETIGEN SIE <Ctrl> UND 7
50 POKE V+21,3                 : REM EINSCHALTEN VON SPRITE 0 UND 1
60 POKE V+28,3                 : REM SPRITE 0 UND 1 SIND MULTI-COLOR
70 POKE V+39,6                 : REM FARBE VON SPRITE 0=BLAU
80 POKE V+40,2                 : REM FARBE VON SPRITE 1=ROT
90 POKE V+37,14                : REM MULTI-COLOR-FARBE 1=HELLBLAU
100 POKE V+38,0                : REM MULTI-COLOR-FARBE 2=SCHWARZ
110 POKE 2040,13               : REM SPRITE 0 AUS BEREICH 832 BIS 895
120 POKE 2041,13               : REM SPRITE 1 EBENSO
130 FOR I=0 TO 62              : REM SCHLEIFE LIEST DATEN EIN
```

```
140 : READ X                  : REM EINLESEN DES WERTES
150 : POKE I+832,X            : REM SPEICHERN DES WERTES
160 NEXT I                    : REM ENDE DER SCHLEIFE
170 POKE V+0,24: POKE V+1,50: REM POSITION VON SPRITE 0
180 POKE V+2,60: POKE V+3,50: REM POSITION VON SPRITE 1
190 FOR D=1 TO 2000: NEXT    : REM WARTESCHLEIFE
200 FOR I=0 TO 500:           : REM SCHLEIFE
210 : POKE V,I AND 255        : REM X-KOORDINATE SPRITE 0
220 : POKE V+2,(500-I) AND 255: REM X-KOORDINATE SPRITE 1
230 : POKE V+1,50+(I AND 7) : REM Y-KOORDINATE SPRITE 0
240 : POKE V+3,50-(I AND 7) : REM Y-KOORDINATE SPRITE 1
250 NEXT
260 GOTO 200: REM NOCHMAL?
1000 DATA 000,000,000
1010 DATA 000,000,000
1020 DATA 000,000,000
1030 DATA 000,000,000
1040 DATA 000,000,000
1050 DATA 000,000,000
1060 DATA 000,000,000
1070 DATA 003,255,255
1080 DATA 000,002,000
1090 DATA 192,170,128
1100 DATA 194,150,080
1110 DATA 192,170,168
1120 DATA 000,032,128
1130 DATA 000,170,160
1140 DATA 000,000,000
1150 DATA 000,000,000
1160 DATA 000,000,000
1170 DATA 000,000,000
1180 DATA 000,000,000
1190 DATA 000,000,000
1200 DATA 000,000,000
```

Sicherlich ist es um einiges komplizierter, Multi-Color-Sprites anzufertigen als Einfarben-Sprites, bei denen ein Punkt auf dem Papier auch einem Punkt auf dem Bildschirm entspricht. Aber es gibt ja zum Glück Sprite-Editoren, die einem die Arbeit erheblich erleichtern. Ein solcher ist auch in BASIC 7.0 eingebaut (*SPRDEF*). Man muß ja nicht immer bei Adam und Eva anfangen. Allerdings, das wurde ja bereits erwähnt, ist es

für die Maschinenprogrammierer unter Ihnen unerläßlich zu wissen, wie die Sprite-Programmierung ohne die komfortablen BASIC-Kommandos funktioniert, dazu die Beispiele.

Die Sprites, die Sie mit dem in BASIC 7.0 eingebauten Sprite-Editor definieren und benutzen, werden im RAM im Speicherbereich $0E00-$1000 abgespeichert.

Übrigens kann man Sprites in jedem der möglichen Modi mit dem Hintergrund überlagern lassen, sei es nun im Text-, Grafik- oder Multi-Color-Grafik-Modus.

2.3.10 Interrupts durch den VIC-Chip

Der VIC-Chip ist auch in der Lage Interrupts auszulösen. *Interrupts* unterbrechen ein laufendes Maschinenprogramm, wenn ein bestimmtes Ereignis eingetreten ist. Es gibt vier verschiedene Interrupt-Quellen beim VIC:

* *Der Lightpen*
* *Der Rasterzeilen-Interrupt*
* *Eine Sprite/Sprite-Kollision*
* *Eine Sprite/Hintergrund-Kollision*

Nicht zuletzt dank der Möglichkeit, Rasterzeilen-Interrupts auslösen zu können, ist es im BASIC 7.0 möglich geworden, Text und Grafik zu mischen (mittels des GRAFIK-Kommandos). Auch hat der Rasterzeilen-Interrupt eine Menge Spiele sehr verfeinert. Wenn Sie einen Interrupt programmieren wollen, so müssen Sie im IMR-Register durch Setzen der entsprechenden Bits angeben, welche Interrupt-Quelle(n) gewünscht ist (sind). Ferner müssen Sie den Interrupt-Vektor auf Ihre eigene Interrupt-Routine verbiegen - damit Sie auch auf den Interrupt reagieren können. Sollte der Interrupt vom CIA1 kommen, so müssen Sie entsprechend in die Kernal-Routine verzweigen. Der CIA1 löst alle 1/60 Sekunden einen Interrupt aus, um die Tastatur abzufragen. Es wird dann ebenfalls in Ihre Interrupt-Routine verzweigt. Durch Abfragen des Registers 13 ICR des CIA1 können Sie feststellen, ob der Interrupt vom CIA1 kam. Sollte der Interrupt vom VIC-Chip kommen, so ist das 7. Bit des

Registers IRR (*Interrupt Request Register*) des VIC-Chips gesetzt, sowie das Bit des Auslösers.

Das Bit des Auslösers brauchen Sie natürlich nur dann abzufragen, wenn mehrere Auslöser in Frage kommen sollten. Wenn Sie lediglich den Rasterzeilen-Interrupt benutzen, so reicht die Abfrage des Bit 7 voll aus. Im IMR (Interrupt Mask Register) können Sie angeben, welche Interrupt-Quellen in Frage kommen. In Register 18 und 17 (Übertrag) können Sie die Rasterzeile programmieren, bei der ein Interrupt ausgelöst werden soll. Wird diese Zeile beim Bildschirmaufbau durchlaufen, so wird ein Interrupt ausgelöst. Wenn die Routine darauf reagiert, befindet sich der Kathodenstrahl allerdings erfahrungsgemäß schon einige Zeilen tiefer. Diese Zeitverzögerung muß man in seine Berechnungen einkalkulieren.

Die Möglichkeiten, die sich durch Interrupt-Programmierung bieten, sowie die Flut an Programmierkniffen zu erwähnen bzw. zu erläutern, würde allerdings an dieser Stelle den Rahmen sprengen. Erlauben Sie uns, Sie an entsprechende Literatur zu verweisen.

2.4 Normale Zeichendarstellung

Unter diesem Begriff dürfen Sie den "normalsten" aller Modi verstehen: die Textdarstellung. Dieser Modus ist auch nach dem Einschalten des Gerätes automatisch gesetzt. Es werden 1000 Zeichen aus dem Video-RAM als Textseite dargestellt; dafür wird Zeichen für Zeichen aus dem Video-RAM gelesen. Jedes Zeichen hat einen Code, der als Zeiger im Zeichengenerator verwendet wird, um das im Zeichengenerator unter diesem Code abgespeicherte Bitmuster an der aktuellen Position darzustellen. Auf diese Art und Weise kann man 256 verschiedene Zeichen auf dem Bildschirm darstellen. Im Commodore 128 sind vier dieser Zeichensätze gespeichert: Sie können mittels Shift/Commodore zwischen Groß-/Kleinschrift und Groß-/Grafikmodus umschalten. Dies sind schon einmal zwei Zeichensätze. Weiterhin können Sie aber noch zwischen Ascii- und DIN-Tastatur auswählen, wodurch noch zwei weitere Zeichensätze verfügbar sind.

Für jedes Zeichen auf dem Bildschirm gibt es noch eine separate Speicherstelle im Farb-RAM, die die Farbe des Zeichens definiert. Bei der Darstellung der Zeichen wird bei gesetztem Bit im Zeichenmuster (das aus dem Zeichengenerator geholt wird) die Farbe aus dem unteren Nibble des Farb-RAMs geholt. Hier können 16 Farben definiert werden. Ist ein Bit nicht gesetzt, so wird die Farbe aus dem Hintergrundfarbregister 0 geholt; der Punkt ist also transparent.

2.4.1 Verschieben des Video-RAMs

Eine nützliche Eigenschaft des VIC-Chips ist die Möglichkeit, Video-RAM und/oder Zeichengenerator zu verschieben. Auf diese Weise kann man beispielsweise zwei oder mehr Textseiten realisieren: Während man die eine anzeigt, kann man die andere schon im Hintergrund aufbauen. Dies gilt in gleicher Weise auch für den Grafikmodus. Allerdings muß man beachten, daß das Farb-RAM unverschiebbar ist.

Wie bereits erwähnt, kann der VIC-Chip lediglich 16 KByte adressieren. Normalerweise werden die ersten 16 KByte der Bank 0 adressiert - das Video-RAM befindet sich an Adresse $0400-07FF. Um dieses Video-RAM in 1-KByte-Schritten zu verschieben, dient das Register 24 des VIC-Chips. Die Bits 4-7 dieses Registers repräsentieren die Adreßbits 10-13 des Video-RAM. Die Adresse $0400 sieht bitweise wie folgt aus:

0000 1000 0000 0000 = $0400

Das ganz linke Bit ist Adreßbit 15, das ganz rechte Bit nennt man Adreßbit 0. Die Adreßbits 10-13 lauten also: 0010. Diese Bitkombination muß sich auch in Register 24, Bits 4-7, befinden. Wollen Sie nun das Video-RAM um ein KByte verschieben, so wäre die Adresse des Video-RAM $0800.

0001 0000 0000 0000 = $0800

Die Adreßbits 10-13 lauten nun 0100. Um beispielsweise diese Adresse in Register 24 zu schreiben, muß man Bits 4-7 erst einmal ausmaskieren (=löschen), dann kann man die Bitkombination durch logische ODER-Verknüpfung definieren:

```
P=PEEK(53248+24): REM Alter Inhalt
POKE 53248+24,(P AND 240) OR 64
```

Dies ist nötig, um die restlichen Bits des Registers nicht zu "stören", da sie die Adresse des Zeichengenerators definieren.

Die Grenze ist also erreicht, wenn man das Video-RAM um mehr als 16 KByte verschieben will. Das Register 24 stellt die Adreßbits 10-13 des Video-RAM zur Verfügung, vollkommen ausreichend für Verschiebungen innerhalb eines Rahmens von 16 KByte. Da im VIC-Chip die Adreßbits 14 und 15 nicht definiert werden können, müssen diese Bits von außerhalb zu Hilfe genommen werden. Diese beiden Bits befinden sich im Register 0 des CIA2 (Adresse $DD00), Bits 0 und 1. Beachten Sie unbedingt, daß diese Bits low-aktiv sind, d.h. daß sich die Werte umkehren. Um die untersten 16 KByte zu adressieren (Adreßbits 14 und 15 sind 0), müssen die Bits 0 und 1 des Registers 0 im CIA2 gesetzt sein.

Wichtig! - Wichtig! - Wichtig!

Wenn Sie die Bits 0 und 1 im CIA2 ändern, so verschiebt sich nicht nur das Video-RAM um 16 KByte, sondern auch auch die Basis des Zeichengenerators. Beachten Sie dies auch besonders bei der Grafik-Programmierung!

Folgende Werte stehen für die jeweiligen Speicherbereiche:

X	Bits	Bereich
0	00	$C000-$FFFF
1	01	$8000-$BFFF
2	10	$4000-$7FFF
3	11	$0000-$3FFF (Einschaltzustand)

```
POKE 56576, A: REM Auswahl der 16-K-Seite
```

2.4.2 Verschieben des Zeichengenerators

Die CIA2-Bits definieren also gleichzeitig die 16-KByte-Seite von Video-RAM und Zeichengenerator. Aber auch den Zeichengenerator kann man verschieben, wenn auch nicht in 1-KByte-Schritten, so doch in 2-KByte-Schritten. Die Bits 1-3 im Register 24 des VIC-Chips repräsentieren die Adreßsbits 11-13 des Zeichengenerators.

Normalerweise zeigt dieser Pointer auf das Character-ROM, welches für die Gestaltung der Zeichen auf dem Bildschirm verantwortlich ist. Gerade im Grafikmodus muß man aber diesen Zeichengenerator verschieben, um die Basis der Grafik zu definieren. (Das Video-RAM wird ja zum Farb-RAM!) Das Character-ROM befindet sich rein physikalisch allerdings außerhalb des Lesebereichs des VIC-Chips, da die Adresse $D000 beispielsweise bei unterster eingeschalteter Seite nicht ansprechbar ist. Dieses Character-ROM besitzt aber dank des Adreß-Managers einen Sonderstatus: Werden die relativen Adressen $1000-$1FFF oder $9000-$9FFF angesprochen, so wird automatisch auf das Character-ROM ($D000-$DFFF) zugegriffen. Sollte Sie dies beispielsweise bei der Programmierung im Grafikmodus stören, so müssen Sie entweder die Seiten 1 oder 3 verwenden oder den Bereich für den Zeichengenerator verschieben.

Wenn Sie beispielsweise ein paar selbstdefinierte Zeichen programmieren und verwenden wollen, so kopieren Sie sich zunächst den Originalzeichensatz aus dem Character-ROM ins RAM. Dann können Sie einzelne Zeichen umdefinieren oder gar ganz neu definieren (z.B. mathematische Symbole?), Sie müssen den VIC-Chip lediglich von Ihrem Vorhaben in Kenntnis setzen und die Adresse angeben, an der sich der neue Zeichensatz befindet.

2.4.3 Das Farb-RAM

Das Farb-RAM ist wohl das einzige, was man beim VIC nicht umdefinieren kann. Das ist aber kein Grund betrübt zu sein, es ist ganz gut, das Farb-RAM immer am gleichen Platz zu wissen. Das Farb-RAM dient bei der Textdarstellung als Farbpalette,

der VIC holt sich hier für jedes Zeichen die entsprechende Farbe. Wenn Sie im Hi-Res-Modus arbeiten, so liegt das Farb-RAM brach. Sie können den Speicher dann anderweitig verwenden. Im Multi-Color-Modus hingegen tritt das Farb-RAM wieder in Aktion - es liefert Farbwerte für ganz bestimmte Bereiche des Bildes (s.u.).

Das Farb-RAM beginnt an der Adresse $D800 und endet an Adresse $D800+999.

2.5 Programmierung von Farbe und Grafik

Wir wollen die "graue Theorie" anhand kleiner Beispiele verdeutlichen.

Immer wenn Sie in die Verlegenheit kommen, eine Farbe zu definieren, sei es nun im Farb-RAM für ein Zeichen auf Ihrem Textschirm oder eine Farbe für ein Sprite, gelten folgende Codes für die jeweiligen Farben:

Taste	*Farbe*	*Kennzahl*
Ctrl-1	*Schwarz*	*0*
Ctrl-2	*Weiß*	*1*
Ctrl-3	*Rot*	*2*
Ctrl-4	*Türkis*	*3*
Ctrl-5	*Violett*	*4*
Ctrl-6	*Grün*	*5*
Ctrl-7	*Blau*	*6*
Ctrl-8	*Gelb*	*7*
C=-1	*Orange*	*8*
C=-2	*Braun*	*9*
C=-3	*Hellrot*	*10*
C=-4	*Grau 1*	*11*
C=-5	*Grau 2*	*12*
C=-6	*Hellgrün*	*13*
C=-7	*Hellblau*	*14*
C=-8	*Grau 3*	*15*

Um beispielsweise den Hintergrund und den Rahmen in Schwarz erscheinen zu lassen, sind folgende Anweisungen notwendig:

```
POKE 53280,0
POKE 53281,0
```

Wir wollen nun den Bildschirm (der jetzt schwarz ist) mit weißen A's auffüllen. Dazu müssen wir das Video-RAM ab Adresse $0400 bis Adresse $0400+999 mit dem Code 1 auffüllen. Zusätzlich müssen wir gleichzeitig das Farb-RAM an Adresse $D800 bis $D800+999 ebenfalls mit 1 (für weiß) belegen:

```
10 PRINT CHR$(147);: REM LÖSCHE BILDSCHIRM
20 FOR I=0 TO 999:    REM 1000 ZEICHEN
30 POKE 55296+I,1:    REM WEISS
40 POKE I+1024,1 :    REM EIN A
50 NEXT I
60 GET A$: IF A$="" THEN 60
```

Die Zeile 60 soll verhindern, daß der Bildschirm gescrollt wird. Bei Tastendruck wird das Programm abgebrochen. Sollte Ihnen dies zu langweilig sein, so probieren Sie folgendes aus:

```
30 POKE 55296+I,RND(0)*16: REM FARBE
40 POKE 1024+I,RND(0)*255:  REM ZEICHEN
```

Was passieren wird, sollen Sie erst beim Ausprobieren sehen. Da die Programmierung des Textbildschirmes aber ebenso einfach wie langweilig ist, wollen wir uns nun der Grafikprogrammierung zuwenden:

2.5.1 Der Hi-Res-Modus

Da wir uns auf der niedrigsten Ebene der Programmierung befinden, haben wir keine Kommandos wie Linie-ziehen oder Kreise-zeichnen - noch nicht einmal ein Kommando zum Setzen eines Punktes existiert. Besonders diejenigen unter Ihnen, die im 64er-Modus programmieren wollen, müssen sich vollkommen von der Vorstellung freimachen, solche Kommandos benutzen zu können. Wenn Sie in Maschinensprache programmieren wollen, so können Sie im 128er-Modus natürlich auf die im ROM

gespeicherten Routinen zurückgreifen. Meist ist es aber vom Zeitaspekt gesehen besser, wenn Sie sich selbst solche Routinen schreiben, da Sie diese Routinen auf Ihren individuellen Bedarf anpassen können. Außerdem machen die Betriebssystemroutinen zeitraubende Abfragen, die in Maschinensprache vollkommen wegfallen können.

Hier ist nun ein Programm, das im Hi-Res-Modus eine Sinuskurve auf den Bildschirm plottet, ohne auch nur einen einzigen Befehl von BASIC 7.0 zu verwenden, es wird alles "zu Fuß" erledigt. Man könnte dieses BASIC-Programm also praktisch direkt in Maschinensprache übersetzen, wobei lediglich die Programmierung der Sinusberechnung einige Probleme darstellen dürfte.

```
10 REM SINUS-PLOT-PROGRAMM
20 V=53248: REM STARTADRESSE DES VIC
30 AD=8192: REM STARTADRESSE DER HI-RES-BIT-MAP
40 POKE V+17,59: REM EINSCHALTEN DER GRAFIK
50 POKE V+24,24: REM DEFINITION DES ZEICHENGENRATORS
60 FOR I=1024 TO 2023: REM SETZEN DES HI-RES-FARB-RAMS
70 : POKE I,16          : REM FARBKENNZAHLEN 1/0
80 NEXT I
90 FOR I=8192 TO 16383: REM LÖSCHEN DER HI-RES-BIT-MAP
100 : POKE I,0
110 NEXT I
120 Y=100: REM POSITION DER X-ACHSE
130 FOR X=0 TO 319: REM ZEICHNEN DER X-ACHSE
140 : GOSUB 1000:   REM PUNKT ZEICHNEN
150 NEXT X
160 X=160: REM POSITION DER Y-ACHSE
170 FOR Y=0 TO 199: REM Y-ACHSE ZEICHNEN
180 : GOSUB 1000
190 NEXT Y
200 X=0
210 FOR I=-3.141592654 TO 3.141592654 STEP 0.0196349541
220 : Y=100+99*SIN(I): REM FUNKTIONSWERT
230 : GOSUB 1000
240 : X=X+1: REM NÄCHSTER FUNKTIONSWERT
250 NEXT I
260 GET A$: IF A$="" THEN 260
270 END
```

```
1000 OY=320*INT(Y/8)+(Y AND 7): REM Y-OFFSET
1010 OX=8*INT(X/8)             : REM X-OFFSET
1020 MA=2^(7-(X AND 7))
1030 AV=AD+OX+OY
1040 POKE AV,PEEK(AV) OR MA: REM SETZEN DES PUNKTES DURCH OR
1050 RETURN
```

Wenn Sie das Programm starten, werden Sie von der Ausführungsgeschwindigkeit nicht gerade begeistert sein. Das liegt an den aufwendigen Berechnungen und den REM-Kommandos. Eine sehr zeitintensive Berechnung ist die 2^-Berechnung, die man sowohl in BASIC als auch in Maschinensprache sehr gut durch eine Tabelle ersetzen kann. Natürlich ließe sich dies alles in BASIC 7.0 viel effektiver und kürzer programmieren, dann aber könnten Sie nicht erkennen, wie das Setzen eines einzelnen Punktes intern funktioniert.

Wir wollen uns nun näher mit dem Programm befassen, um herauszufinden, wie wir die Grafik auf den Bildschirm gezaubert haben.

Um uns die Berechnungen im Programm bezüglich des VIC-Chips zu erleichtern, haben wir einmal die Startadresse des Chips definiert. Außerdem kann man dann auch deutlicher erkennen, welche Register angesprochen werden. Als erstes wird das Register 17 geändert, es wird der Wert 59 hineingeschrieben. Hauptsächlich wird allerdings das Bit 5 gesetzt, um VIC mitzuteilen, daß wir uns nun im Grafikmodus befinden. Im Register 24 werden die Startadressen von Video-RAM und Zeichengenerator abgelegt. Wir schreiben in dieses Register eine 24.

24 = $18 = %0001 1000

Die Bits 4-7 des Registers bestimmen die Adreßregister 10-13 des Video-RAM (s.o.) - wir erhalten also als Startadresse $0400, den Normalwert des Bildschirmes. Weiterhin bestimmen die Bits 1-3 die Adreßbits 11-13 der Zeichenbasis:

%0010 0000 0000 0000 = $2000 = 8192

Wir haben also mit einem POKE-Kommando sowohl die Adresse des Video-RAM als auch die Adresse der Bit-Map definiert. Erfahrungsgemäß schleichen sich bei der Umrechnung dieser zwei Adressen die meisten Fehler ein. Darum sollte man sich ähnlich ausführlich wie in unserem Beispiel die zwei Adressen aufschreiben und die Bits, die benötigt werden, unterstreichen und dann zusammenfassen.

Wenn Sie das Programm starten, so kommen Sie durch Betätigen einer Taste zwar wieder ins BASIC zurück; Sie sehen aber nichts, da der Grafikmodus nicht ausgeschaltet wird - Sie erkennen lediglich, daß Ihre Kurve recht farbig wird. Dies liegt daran, daß das Video-RAM nun mit Werten gefüllt wird, die diese Farben hervorrufen. Darum sollte man, bevor die Register 17 und 24 überschrieben werden, die Inhalte dieser Register zwischenspeichern, damit man sie später wieder rekonstruieren kann. Fügen Sie folgende Zeilen ein:

```
35 A1=PEEK(V+17): A2=PEEK(V+24)
270 POKE V+17, A1: POKE V+24, A2: END
```

Jetzt wird nach Betätigen einer Taste wieder in den Textmodus zurückgeschaltet.

Wir befinden uns also im sogenannten HI-RES-MODUS, in dem wir eine Auflösung von 320 x 200 Punkten haben. Das sind genau 64.000 Punkte, über die wir frei verfügen können. Da 8 Punkte = 8 Bits zu je einem Byte zusammengefaßt werden, benötigen wir einen Speicherplatz von exakt 8.000 Bytes, um unsere Grafik darzustellen. In einer Zeile lassen sich 320 Punkte darstellen, also 40 Bytes (320/8), so wie wir es aus dem Textmodus kennen. Weiterhin verfügen wir über 25 Zeilen zu je 8 Punkten. Sie bemerken sicherlich schon die Parallelen zum Textmodus!

Ein Zeichen im Textmodus besteht aus 8 x 8 = 64 Punkten, die unabhängig voneinander gesetzt oder gelöscht werden können. Die Farbe für die gesetzten Punkte wird dem Farb-RAM entnommen, die Farbe für die nicht gesetzten Punkte wird dem Hintergrundfarbregister 0 entnommen. Ähnlich verhält sich dies im Grafikmodus. Es werden auch hier immer 8 x 8 Punkte zu einer Einheit zusammengefaßt. In diesem Kästchen von 64

Punkten kann man dann zwei Farben darstellen lassen. Würde man für jedes Bit extra einen Speicherplatz vorsehen, der die Farbe definiert, so bräuchte man allein für die Farbdefinition 64 KByte! Durch das Zusammenfassen von je 8 x 8 Punkten benötigt man lediglich einen Speicherplatz von 1.000 Bytes für die Farbdefinition. Wir wollen nun einmal eine solche 8x8-Einheit näher betrachten:

```
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
```

Eine solche Einheit nennt man auch *Matrix* oder *Zeichenmatrix*. In dieser Matrix sind also auch alle unsere Buchstaben und Sonderzeichen definiert, die wir im Textmodus auf dem Bildschirm sehen können. Im Hi-Res-Modus können wir jetzt alle Matrizen selbst bestimmen und haben nicht mehr nur eine "Zeigertabelle" auf fertige Matrizen (Zeichengenerator) - etwas anderes ist unser Textspeicher ja nicht. Das klingt jetzt vielleicht alles noch ein wenig kompliziert, ist es aber eigentlich nicht.

Sie sehen, daß es ohne größeren Programmieraufwand auch möglich sein muß, Text und Grafik zu mischen bzw. den Text in die restliche Grafiklandschaft zu "malen". In den Grafikspeicher hineinzuschreiben, geht natürlich nicht. Wie wird aber jetzt die Grafik genau auf dem Bildschirm gebracht? Welche Speicherstelle in unserem Grafikspeicher definiert welche 8 Punkte in unserer Grafik? Diese Fragen soll folgendes Schaubild verdeutlichen:

8192: . . . . *8200:* . . . .
8193: . . . . *8201:* . . . .
8194: . . . . *8202:* . . . .
8195: . . . . *8203:* . . . .
8196: . . . . *8204:* . . . .
8197: . . . . *8205:* . . . .
8198: . . . . *8206:* . . . .
8199: . . . . *8207:* . . . .

8224: . . . . *8527:* . . . . *usw.*

Dieses Schaubild ist vollkommen ausreichend, um sowohl den Wechsel zwischen Spalten und den Wechsel zwischen Zeilen zu verdeutlichen, zumindest, was die Adressierumg betrifft. Jeder Punkt repräsentiert ein Bit. Unser Grafikspeicher beginnt an Adresse 8192 und definiert mit seinem ersten Byte die ersten 8 Punkte unserer Grafik. Wollten wir allerdings den neunten Punkt in der ersten Zeile ansprechen, so müßten wir uns an die Adresse 8200 wenden, dort *wohnt* dieser Punkt nämlich. Sie erkennen sicherlich das Schema der Darstellung: Es wird wie im Textmodus verfahren, es wird Zeichen für Zeichen und Zeile für Zeile dargestellt. Aber wie spricht man nun einen ganz bestimmten Punkt an? Wir müssen zuerst die Adresse berechnen, in der er *haust*. Um einen solchen Algorithmus aufzustellen, beginnt man meist, indem man die Bedingungen vereinfacht. Wir wollen also zuerst einmal nur einen Punkt in der ersten Zeile ansprechen:

```
AD=8192 + INT(X/8)
```

Wir wollen den Term INT(X / 8) * 8 der Einfachheit halber OX = Offset der X-Position nennen. Das ist schon alles, was wir bezüglich der X-Koordinate bedenken müssen. Wir haben also jetzt die Adresse des Punktes, allerdings wissen wir noch nicht, welches Bit angesprochen wird. Wir wollen ja keinen der Nachbarn wecken:

```
BIT = X - INT(X / 8) * 8
```

Wir müssen also den Rest von X / 8 ermitteln. Dies geht aber auch einfacher, indem man durch logisches UND-Verknüpfen die untersten 3 Bits ausmaskiert.

```
BIT = X AND 7
```

Probieren Sie es ruhig aus, es funktioniert und ist viel schneller als die Division, besonders in Maschinensprache. Jetzt müssen wir aber noch bedenken, daß das am weitesten links wohnende Bit nicht den Namen *0* trägt (was ja die Restfunktion ergäbe), sondern den Namen *7*. Entsprechend verhält sich das mit dem ganz rechts wohnenden Bit. Wir müssen dieses Verhältnis also noch umkehren:

```
BIT = 7 - (X AND 7)
```

Jetzt stimmt die Anrede. Um einen solchen Punkt in BASIC oder Assembler zu setzen, müssen wir die angesprochene Speicherstelle mit logischer ODER-Verknüpfung setzen. Dazu müssen wir die Zweier-Potenz errechnen, also

```
2^(7-(X AND 7))
```

Jetzt können wir schon jeden beliebigen Punkt in der ersten Zeile setzen:

```
POKE 8192+OX,PEEK(8192+OX) OR 2^(7-(X AND 7))
```

Um die ersten acht Zeilen anzusprechen, brauchen wir lediglich die Y-Koordinate hinzuzuaddieren. Wenn wir die neunte Zeile ansprechen wollen, so müssen wir 320 Bytes überspringen. Daraus ergibt sich folgende Addition bezüglich der Y-Position:

```
OY=INT(Y/8)*320 + (Y AND 7)
```

Um nun einen Punkt anzusprechen, müssen wir den Offset von X- und Y-Position zu der Basisadresse des Grafikspeichers hinzuaddieren. Es ergibt sich also für die Adressenberechnung folgende Formel:

```
AD = OX + OY + 8192
```

In der Formel sind unsere Terme zur Berechnung des X- und Y-Offsets integriert. Um einen beliebigen Punkt zu setzen, haben wir nun also alle nötigen Berechnungen hergeleitet. Es ergibt sich (in BASIC) die folgende Kommandofolge:

```
OY = 320*INT(Y/8) + (Y AND 7)
OX = 8 *INT(X/8)
BI = 2^(X AND 7)
AD = 8192 + OX + OY
POKE AV, PEEK(AV) OR BI
```

Dies ist jetzt sehr ausführlich. Wollen wir einen Punkt löschen, so ändert sich natürlich an der Adressenberechnung nichts, lediglich das POKE-Kommando müssen wir ändern:

```
POKE AV, PEEK(AV) AND NOT BI
```

Wir maskieren also das errechnete Bit aus; das ist alles, was wir zu tun haben.

Wir wissen nun, wie wir Punkte setzen und wieder löschen können. Allerdings wissen wir noch nicht, wie wir die Farben bestimmen können, die bei gesetztem und gelöschtem Bit angezeigt werden sollen. In unserem Beispielprogramm befindet sich die Bit-Map (praktisch der Zeichengenerator) an der Adresse 8192 bis 16192. Sie erinnern sich bestimmt auch noch, daß wir das normale Video-RAM zum Farb-RAM degradiert haben. Das heißt nichts anderes, als daß die Informationen zur Farbgestaltung unserer Grafik aus genau diesem Speicher geholt werden, in dem sich sonst der Inhalt des Bildschirmes befindet. Dieser Speicher befindet sich an der Adresse 1024 bis 2023.

Da wir mit einem Bit zwei Farben definieren können, müssen wir diese zwei Farben auch im Video-RAM unterbringen können. Sie erinnern sich bestimmt noch an den Aufbau des Grafikbildschirmes. Wir hatten immer *Päckchen* zu 8 Byte - acht aufeinanderfolgende Byte in unserer Bit-Map. Ein solches *Päckchen* hat also auch genau die Größe eines normalen Zeichens auf dem Bildschirm. Die Farben für unser erstes Päckchen an der Adresse 8192 - 8199 werden im ersten Byte des Video-RAM definiert - also an der Adresse 1024. Für alle 64 Punkte in diesem Päckchen gelten also diese beiden Farben. Für

das zweite Päckchen von Adresse 8200 bis 8207 werden die Farben entsprechend in Adresse 1025 festgelegt. Die Frage bleibt: Wie werden die Farben definiert?

Dazu sehen wir einmal in unserem Beispielprogramm nach, in dem wir mittels einer FOR-NEXT-Schleife den Bereich von 1024 bis 2023 mit dem Wert 16 aufgefüllt haben. Wie sieht die Zahl 16 in Binärschreibweise aus?

16 = $10 = %00010000

Wenn wir das obere und das untere Nibble (Einheit von vier Bits) voneinander trennen, so erhalten wir zwei Werte zwischen 0 und 15 - ausreichend also, um die verfügbaren Farben zu definieren. In diesem Beispiel erhalten wir die Werte 1 und 0. Wenn wir in der Farbtabelle nachsehen, so stellen wir fest, daß wir die Farben Weiß und Schwarz definiert haben. Gerade im HI-RES-Modus muß man mit sehr viel Vorbehalt die Farben definieren, damit man auch genügend Kontrast erhält. Nicht selten muß man zwei nebeneinanderliegende Punkte setzen, damit man überhaupt die Farbe erkennen kann. Dies ist aber von Monitor zu Monitor verschieden. Der Kontrast zwischen Weiß und Schwarz ist sicherlich einer der besten (umgekehrt vielleicht sogar noch besser!), Rot und Blau beispielsweise sieht hingegen chaotisch aus. Die Farbe, die im oberen Nibble des Farb-RAMs definiert wird, wird bei gesetztem Bit angezeigt - in unserem Beispiel bedeutet das, daß der Hintergrund schwarz (0) ist und mit weiß (1) gemalt wird. Um das Farb-RAM zu setzen, gilt folgende Regel:

POKE <Farbram>,<Vordergrund>*16 + <Hintergrund>

Natürlich können Sie über den gesamten Grafikbildschirm verstreut weit mehr als zwei Farben definieren: Es gibt 256 mögliche Kombinationen innerhalb eines Päckchens, Schwarz-Weiß ist nur eine davon. Die Programmierung im HI-RES-Modus erlernt man am besten, indem man sich einfach der Methode *Try-And-Error* bedient, d.h. probieren und Fehler erkennen.

2.5.2 Der Multi-Color-Modus

Neben dem Hi-Res-Modus gibt es noch eine zweite Möglichkeit, Grafiken auf dem Bildschirm darzustellen: Es ist der sogenannte Multi-Color-Modus. Wir kennen das Wort Multi-Color ja bereits von den Sprites. Im Multi-Color-Modus verfügen Sie über 4 Farben pro Päckchen - allerdings zu Lasten der Auflösung: Die Auflösung im Multi-Color-Modus beträgt "nur" noch 160 x 200 Punkte - also exakt die Hälfte. In einem Byte werden jetzt auch nicht mehr acht Punkte definiert, sondern lediglich noch vier. Um den Multi-Color-Modus einzuschalten, muß man im Register 17 das Bit 5 setzen (genauso, wie im Hi-Res-Modus). Zusätzlich muß im Register 22 das 4. Bit gesetzt werden. Dies geschieht durch die Anweisung:

```
POKE 53248+22,PEEK(53248+22) OR 16
```

Die Adressen für Bit-Map und Farb-RAM werden auf dieselbe Weise programmiert wie im Hi-Res-Modus. In Adresse 8192 (das erste Byte in der Bit-Map) soll sich folgender Inhalt befinden:

```
PEEK(8192)= %00011011 = $1B = 27
```

In diesem Byte werden also die ersten vier Punkte der ersten Zeile definiert. Da je zwei Bits zusammengefaßt werden, erhalten wir die Bitpaare 00, 01, 10 und 11 - alle vier möglichen Kombinationen also.

Bits	Farbinformation kommt von
00	Hintergrundfarbregister 0
01	Oberen vier Bits des Video-RAM
10	Unteren vier Bits des Video-RAM
11	Farb-RAM

Hier ist lediglich die Bitkombination 00 auf dem gesamten Grafikbildschirm gleich. Die Bitkombinationen 01 und 10 verhalten sich genauso, wie für dem Hi-Res-Modus beschrieben. Das Farb-RAM beginnt an Adresse $D800 und stellt je eine Farbe zur Verfügung. Die Programmierung im Multi-Color-Modus empfiehlt sich eigentlich in den meisten Fällen, da die Farbenvielfalt eine Menge interessanter Möglichkeiten bietet.

Natürlich muß man bei der Berechnung der Adressen bedenken, daß pro Byte lediglich vier Punkte definiert werden. Es ändert sich also die Formel für den X-Offset, sie muß im MULTI-COLOR-Modus wie folgt lauten:

```
OX = 8 * INT(X/4)
MA = 2^(6-2*(X AND 3))
POKE AV, PEEK(AV) OR MA*<Bitmuster>
```

Sie sehen, daß sich auch die Formel für die Bitbestimmung etwas gewandelt hat. Man muß bedenken, daß man ein Bitpaar mit dem vorhandenen Inhalt logisch ODER-verknüpfen muß und somit die Zweierpotenzen nur in Zweierschritten durchgehen darf. Das *<Bitmuster>* wird durch die Multiplikation mit *MA* dann entsprechend weit nach links geschoben, in der Maschinensprache sagt man geshifted. Da der Multi-Color-Modus der wohl meist verwandte gerade bei Spielen ist, sollte man sich mit den Programmierkniffen in diesem Modus besonders vertraut machen.

2.5.3 Der Multi-Color-Modus (Text)

(Register 22 Bit 4=1)
Weitesgehend unbekannt, aber dennoch möglich ist der Multi-Color-Modus des Textbildschirmes. In diesem Modus können die Zeichen auf dem Bildschirm ebenfalls in mehr als einer Farbe erscheinen. Beispielsweise könnte man eine *0* definieren, die aus einem weißen Kreis mit blauem Querstrich besteht. Ist der Multi-Color-Modus eingeschaltet, dann überprüft VIC, ob Bit 3 des Farbregisters gesetzt ist. Das bedeutet nichts anderes, als daß die Farbe für das entsprechende Zeichen größer als 7 ist (8 - 15). Ist dies der Fall, so wird das Zeichen im Multi-Color-Modus dargestellt. Das Zeichen hat nun keine 8 x 8-Matrix mehr, sondern lediglich eine 4 x 8-Matrix mit folgenden möglichen Bitkombinationen:

Bits	Farbregister	Wird an Adresse X definiert
00	Hintergrundfarbe 0	$D021 (53281)
01	Hintergrundfarbe 1	$D022 (53282)
10	Hintergrundfarbe 2	$D023 (53283)
11	Durch Farbregister	Farb-RAM $D800-$D800+1000

Sollte die Bitkombination 11 lauten, so wird die Farbe aus den unteren drei Bits des Farbregisters ermittelt. Sollte Bit 3 im Farbregister nicht gesetzt sein (Farbe 0-7), so wird eine normale einfarbige 8x8-Matrix dargestellt. Dieser Modus erhält nur dann einen tiefergreifenden Sinn, wenn man sich selber einen Zeichensatz definiert. In einigen Spielen wird von diesem Modus Gebrauch gemacht, da er leichter zu programmieren ist als der Hi-Res-Modus. Schalten Sie ruhig einmal in diesen Modus: Da diese Zeichen nicht auf den Multi-Color-Modus ausgelegt sind, ergibt sich ein farbiges Spektakel:

```
POKE 53248+22,PEEK(53248+22) OR 16
```

Um diesen Modus wieder abzuschalten, reicht das Kommando:

```
POKE 53248+22,PEEK(53248+22) AND 239
```

2.5.4 Extended-Color-Modus

(Register 17 Bit 6=1)
Doch das reichte den Herstellern des VIC-Chips immer noch nicht. Man kreierte noch einen weiteren Modus: den EXTENDED-COLOR-Modus. Dieser Modus ähnelt dem normalen Textmodus sehr. Auch hier kann ein Zeichen aus nur zwei Farben bestehen, jedoch ist die Hintergrundfarbe nicht notwendigerweise immer die gleiche. Man kann unter drei verschiedenen Hintergrundfarben wählen (für die 0-Bits), die 1-Bits beziehen ihre Farbe aus dem Farbregister. Die Hintergrundfarbe wird aus den höchstwertigen zwei Bits aus dem Video-RAM bestimmt:

Bits	Hintergrundfarbregister #
00	0
01	1
10	2
11	3

Da im Video-RAM zwei Bits abgezwackt werden, verbleiben nur noch sechs Bits, um das darzustellende Zeichen zu definieren. Daraus wiederum folgt, daß wir nur noch 64 Zeichen darstellen können - dies sind die untersten 64 Zeichen. So hat jede Medaille zwei Seiten...

2.6 Soft-Scrolling (Smooth Scrolling)

Vielleicht hat der eine oder andere von Ihnen dieses Wort schon einmal in der Fachpresse oder -literatur gelesen und sich gefragt, was man darunter zu verstehen hat.

Soft-Scrolling ist so schön, wie es klingt: Mittels dieser Möglichkeit kann man den Bildschirm horizontal oder vertikal um einen Pixel verschieben. *Scrollen* nennt man das Verschieben des Bildschirmes; ein *Pixel* ist ein anderes Wort für *Punkt*. Dadurch kann man dann, wieder einmal besonders bei Spielen, gleitende Bewegungsabläufe erreichen, so daß man ein *weiches* Scrolling erhält. (Also tatsächlich soft!) Dieses Verschieben kann in eine der vier Himmelsrichtungen (rechts, links, oben oder unten) erfolgen. Beim Verschieben in eine dieser Richtungen wird eine Punktreihe abgedeckt, dafür erscheint am anderen Ende eine Punktreihe neu auf dem Bildschirm. Durch dieses Scrollen kann man den Bildschirm in acht verschiedene Positionen bringen, also vollkommen ausreichend, um ein Zeichen langsam erscheinen zu lassen. Wenn man vom Soft-Scrolling Gebrauch machen will, so muß man den Bildschirm verkleinern. Der VIC-Chip stellt einem hierzu zwei Bits zur Verfügung, in denen man angeben kann, ob man einen 38/40-Zeichen/Zeile-Modus wünscht bzw. einen 24/25-Zeilen-Modus. Der Rahmen vergrößert sich dann entsprechend. Wollen wir vertikal verschieben, so müssen wir eine Zeile opfern, wollen wir hingegen horizontal scrollen, so fallen zwei Zeichen pro Zeile weg.

Um in den 38-Zeichen-Modus zu schalten, muß das Bit 3 des Registers 22 rückgesetzt werden:

POKE 53248+22,PEEK(53248+22) AND 247

Nachdem Sie diese Zeile eingegeben haben, verkleinert sich der eigentliche Bildschirm. Um wieder auf *Normalmodus* zu schalten, müssen wir das Bit 3 wieder setzen:

POKE 53248+22,PEEK(53248+22) OR 8

Dasselbe gilt für den 24-Zeilen-Modus, hier muß das Bit 3 im Register 17 rückgesetzt werden, wenn wir 24 Zeilen haben wollen:

POKE 53248+17,PEEK(53248+17) AND 247 sowie
POKE 53248+17,PEEK(53248+17) OR 8

Im Register 22 wird schließlich in den Bits 0-2 angegeben, welchen Offset der linke Bildrand besitzt. Durch Variieren dieser drei Bits erreicht man das eigentliche Soft-Scrolling in horizontaler Richtung. Will man vertikal scrollen, so muß man den Offset im Register 17 entsprechend ändern.

Doch wir wollen Sie nicht länger auf die Folter spannen. Hier ist ein Demo-Programm, um zu veranschaulichen, welchen Effekt man durch Soft-Scrolling erzielt.

```
10 PRINT CHR$(147) : REM CLR SCREEN
20 POKE 53248+17,PEEK(53248+17) AND 247
30 FOR I=1 TO 24
40 : PRINT "     HALLO !!"
50 NEXT I: PRINT "     HALLO !!"; : REM KEIN SCROLLEN
60 POKE  53248+17,PEEK(53248+17) AND 248 OR 7 : REM SETZE ERSTE PO-
         SITION
70 FOR I=6 TO 0 STEP -1
80 POKE 53248+17,PEEK(53248+17) AND 248 OR P
90 FOR I1=1 TO 60: NEXT I1: REM WARTESCHLEIFE
100 NEXT:                   REM SCHLEIFENENDE
110 GOTO 60:                REM NOCHMAL
```

Natürlich funktioniert dieses Soft-Scrolling auch im Grafikmodus. Gerade hier lassen sich die raffiniertesten Effekte erzielen - beispielsweise kann man ein Raumschiff lautlos durch das unendliche Weltall gleiten lassen. Nachdem man alle 8 Punktreihen durchgescrollt hat, muß man dann eine Grafikspalte oder -zeile mit neuen Werten auffüllen.

Sie sehen, was der VIC-II-Chip Ihnen so alles bietet. Nicht alles wird von den BASIC-7.0-Kommandos abgedeckt, so daß einiges an Möglichkeiten trockengelegt wurde. Diese Möglichkeiten aber wurden in diesem Kapitel erwähnt und die Anwendung näher erläutert, so daß Sie auf nichts, was der VIC-II-Chip in seiner Trickkiste verbirgt, in Zukunft verzichten müssen.

3. Ein- und Ausgabesteuerung

3.1 Allgemeines über den CIA 6526

CIA steht für *Central Intelligence Agency*, doch das gehört hier eigentlich weniger hin. *CIA* steht nämlich auch für *Complex Interface Adapter*, und das sollte uns dann schon mehr interessieren. Es handelt sich im Commodore 128 um den CIA 6526. Seine Leistungsmerkmale in Stichpunkten:

* *16 einzeln programmierbare Ein-/Ausgabeleitungen*
* *8- oder 16-Bit-Handshake sowohl bei der Ein- als auch bei der Ausgabe*
* *2 unabhängige kaskadierbare 16-Bit-Intervalltimer*
* *24-Stunden-(AM/PM)-Uhr mit programmierbarer Alarmzeit*
* *8-Bit-Schieberegister für die serielle Ein-/Ausgabe*

3.1.1 Pinbelegung des 40poligen Gehäuses:

1	Masse
2-9	I/O-Port A; 8 Bit direktional
10-17	I/O-Port B; 8 Bit direktional Die Bits 6 und 7 können zur Anzeige des Unterlaufs der beiden Timer programmiert werden.
18	-PC(Port Control); nur Ausgang; signalisiert die Verfügbarkeit von Daten am Port B oder an beiden Ports.
19	TOD(Time of Day); nur Eingang 50/60 Hz; triggert die Echtzeituhr
20	+5V; Betriebsspannung
21	-IRQ(Interrupt Request); nur Ausgang; wird 0 bei Übereinstimmung eines gesetzten Bits im ICR mit dem Eintreffen eines zugehörigen Ereignisses.
22	R/W(Read-Write); nur Eingang; 0=Übernahme vom Datenbus 1=Übergabe auf Datenbus
23	-CS(Chip Select); nur Eingang; 0=Datenbus gültig, 1=Datenbus hochohmig(Tri-State)
24	-FLAG; nur Eingang; Bedeutung wie -PC
25	02(Systemtakt 2); nur Eingang; alle Datenbusaktionen finden nur bei 02=1 statt.

26-33	DB7-DB0(Datenbus); bidirektional; Schnittstelle zum Prozessor
34	-RES(Reset); nur Eingang; 0=Rücksetzen des CIA in den Grundzustand
35-38	RS3 - RS0(Register Select); nur Eingang; dient der Auswahl eines der 16-Bit-Register; nur gültig mit -CS=0
39	SP(Serial Port); bidirektional; dient als Eingang/Ausgang des Schieberegisters
40	CNT(Count); bidirektional; Eingang/Ausgang des Schieberegistertaktes oder Triggereingang für die Intervalltimer

3.2 Registerbeschreibung der CIA

REG 0 PRA (Port Register A)
Zugriff: Read/Write
Bits 0-7: Dieses Register entspricht dem Zustand der Pins PA0-7.

REG 1 PRB (Port Register B)
Zugriff: Read/Write
Bits 0-7: Dieses Register entspricht dem Zustand der Pins PB0-7.

REG 2 DDRA (Datenrichtung Register A)
Zugriff: Read/Write
Bits 0-7: Diese Bits bestimmen die Datenrichtung der korrespondierenden Datenbits des Ports A.
0=Eingang, 1=Ausgang

REG 3 DDRB (Datenrichtung Register B)
Zugriff: Read/Write
Bits 0-7: Diese Bits bestimmen die Datenrichtung der entsprechenden Datenbits des Ports B.
0=Eingang, 1=Ausgang

REG 4 TA LO (Timer A, Low-Byte)
Zugriff: Read
Bits 0-7: Dieses Register gibt den augenblicklichen Zustand des niederwertigen Bytes von Timer A wieder.
Zugriff: Write
Bits 0-7: In dieses Register wird das niederwertige Byte des Wertes geladen, von dem der Timer bis auf Null zählen soll.

REG 5 TA HI (Timer A, High-Byte)
Zugriff: Read
Bits 0-7: Dieses Register gibt den augenblicklichen Zustand des höherwertigen Bytes von Timer A wieder.
Zugriff: Write
Bits 0-7: In dieses Register wird das höherwertige Byte des Wertes geladen, von dem der Timer bis auf Null zählen soll.

REG 6 TB LO (Timer B, Low-Byte)
Siehe Register 4!

REG 7 TB HI (Timer B, High-Byte)
Siehe Register 5!

REG 8 TOD 10THS (Uhr 1/10 Sekunden)
Zugriff: Read
Bits 0-3: Zehntelsekunden der Echtzeituhr im BCD- Format
Bits 4-7: Immer 0
Zugriff: Write und CRB Bit 7=0
Bits 0-3: Zehntelsekunden im BCD-Format
Bits 4-7: Müssen Null sein!
Zugriff: Write und CRB Bit 7=1
Bits 0-3: Vorwahl der Zehntelsekunden der Alarm zeit im BCD-Format
Bits 4-7: Müssen Null sein!

REG 9 TOD SEC (Uhr Sekunden)
Zugriff: Read
Bits 0-3: Einersekunden im BCD-Format
Bits 4-6: Zehnersekunden im BCD-Format
Bit 7: Immer Null
Weitere Zugriffsarten analog zu REG 8.

REG 10 TOD MIN (Uhr Minuten)
Zugriff: Read
- Bits 0-3: Einerminuten im BCD-Format
- Bits 4-6: Zehnerminuten im BCD-Format
- Bit 7: Immer Null

Weitere Zugriffsarten analog zu REG 8.

REG 11 TOD HR (Uhr Stunden)
Zugriff: Read
- Bits 0-3: Einerstunden im BCD-Format
- Bit 4: Zehnerstunde der Uhr
- Bits 5-6: Immer Null
- Bit 7: 0= Vormittag (AM), 1= Nachmittag (PM)

Weitere Zugriffsarten analog zu REG 8.

REG 12 SDR (Serial Data Register)
Zugriff: Read/Write
- Bits 0-7: Aus diesem Register werden die Daten bitweise zum Pin SP hinausgeschoben bzw. vom Pin SP in dieses Register hineingeschoben.

REG 13 ICR (Interrupt Control Register)
Zugriff: Read (INT DATA)
- Bit 0: 1= Unterlauf Timer A
- Bit 1: 1= Unterlauf Timer B
- Bit 2: 1= Gleichheit von Uhrzeit und gewählter Alarmzeit
- Bit 3: 1= SDR voll/leer (abhängig von der Betriebsart)
- Bit 4: 1= Signal am Pin FLAG aufgetreten
- Bits 5-6: Immer Null
- Bit 7: Übereinstimmung mindestens eines Bits von INT MASK und INT DATA aufgetreten

Achtung: Beim Lesen dieses Registers werden alle Bits gelöscht!

Zugriff: Write (INT MASK)
Bedeutung der Bits wie oben, ausgenommen Bit 7:
- Bit 7: 1= Jedes 1-Bit setzt das korrespondierende Maskenbit. Die anderen bleiben unberührt. 0= Jedes 1-Bit löscht das korrespondierende Maskenbit. Die anderen bleiben unberührt.

REG 14 CRA (Control Register A)
Zugriff: Read/Write
- Bit 0: 1= Timer A Start, 0= Stop
- Bit 1: 1= Unterlauf von Timer A wird an Pin PB6 signalisiert.

Bit 2: 1=Jeder Unterlauf von Timer A kippt PB6 in die jeweils andere Lage.
0= Jeder Unterlauf von Timer A erzeugt an PB6 einen HI-Impuls mit der Länge eines Systemtaktes.

Bit 3: 1= Timer A zählt nur einmal vom Ausgangswert auf Null und hält dann an.
0= Timer A zählt fortlaufend vom Ausgangswert auf null.

Bit 4: 1= Unbedingtes Laden eines neuen Startwertes in Timer A. Dieses Bit fungiert als Strobe. Es muß bei jedem unbedingten Laden neu gesetzt werden.

Bit 5: Dieses Bit bestimmt die Quelle des Timer-Triggers.
1= Timer zählt steigende CNT-Flanken, 0= Timer zählt Systemtaktpulse.

Bit 6: 1=SP ist Ausgang, 0=SP ist Eingang.

Bit 7: 1=Echtzeituhr-Trigger beträgt 50Hz,
0= Echtzeituhr-Trigger beträgt 60Hz.

REG 15 CRB (Control Register B)
Zugriff: Read/Write

Bits 0-4: Diese Bits haben die gleiche Bedeutung wie in REG 14, allerdings bezogen auf Timer B und Pin PB7.

Bits 5-6: Diese Bits bestimmen die Quelle des Triggers für Timer B. 00= Timer zählt Systemtakte, 01= Timer zählt steigende CNT-Flanken, 10= Timer zählt Unterläufe von Timer A, 11= Timer zählt Unterläufe von Timer A, wenn CNT=1 ist.

Bit 7: 1=Alarm setzen, 0=Uhrzeit setzen.

3.3 E/A-Ports

Die Ports A und B bestehen je aus einem 8-Bit-Datenregister *PR* und einem 8-Bit-Datenrichtungsregister *DDR*. Wenn ein Bit im DDR gesetzt ist, arbeitet das korrespondierende Bit im PR als Ausgang. Ist ein Bit im DDR=0, so ist das entsprechende Bit im PR als Eingang definiert.

Während des Lesezugriffs gibt das PR den augenblicklichen Zustand der entsprechenden Pins (PA0-7, PB0-7) wieder; dies sowohl für die Eingangs- als auch für die Ausgangsbits. Darüber hinaus können PB6 und PB7 noch Ausgangsfunktionen für die beiden Timer übernehmen.

Der Datentransfer zwischen der CIA und der an PA/PB angeschlossenen "Außenwelt" kann durch einen Quittungsbetrieb erreicht werden. Hierzu dienen PC und FLAG.

PC wird für die Dauer eines Taktes 0, wenn ein Lese- oder Schreibzugriff auf PRB vorangegangen ist. Dieses Signal kann so die Verfügbarkeit von Daten an PB bzw. die Annahme von Daten von PB anzeigen. FLAG ist ein negativ flankengetriggerter Eingang, der beispielsweise mit PC einer anderen CIA verbunden werden könnte. Eine fallende Flanke an FLAG setzt auch das FLAG-Interrupt-Bit.

Der serielle Datenport SDR ist ein synchrones 8-Bit-Schieberegister. Bit 6 der CRA bestimmt Ein- oder Ausgabemodus. Im Eingabemodus werden die Daten an SP mit der steigenden Flanke eines an CNT liegenden Signales in ein Schieberegister übernommen. Nach 8 CNT-Pulsen wird der Inhalt des Schieberegisters nach SDR gebracht und das SP-Bit im ICR gesetzt.

Im Ausgabemodus fungiert Timer A als Baudratengenerator. Die Daten aus SDR werden mit der halben Unterlauffrequenz von Timer A nach SP hinausgeschoben. Die theoretisch höchste Baudrate beträgt demnach 1/4 des Systemtaktes.

Die Übertragung beginnt, nachdem Daten ins SDR geschrieben wurden, vorausgesetzt Timer A läuft und befindet sich im Continous-Modus (CRA Bit 0=1 und Bit 3=0). Der von Timer A abgeleitete Takt erscheint an CNT. Die Daten aus SDR werden in das Schieberegister geladen und dann mit jeder fallenden Flanke an CNT aus SP hinausgeschoben.

Nach 8 CNT-Impulsen wird der SP-Interrupt erzeugt. Wird jedoch SDR vor diesem Ereignis mit neuen Daten geladen, so werden diese nun automatisch ins Schieberegister geladen und hinausgeschoben. In diesem Falle wird kein Interrupt ausgelöst. Die Daten aus SDR werden mit dem höchstwertigen Bit voran hinausgeschoben. Eingehende Daten sollten dasselbe Format aufweisen.

3.4 Die Timer

Beide Timer haben einen 16-Bit-Zähler *(Read-Only)* und einen 16-Bit-Zwischenspeicher *(Write-Only)*. Wird ein Timer ausgelesen, so wird der augenblickliche Zustand des Zählers wiedergegeben. Beim Schreiben werden die Daten erst im Zwischenspeicher geladen. Die beiden Timer können sowohl unabhängig voneinander als auch im Zusammenhang benutzt werden. Die verschiedenen Betriebsarten erlauben lange Zeitverzögerungen, variable Pulslängen und Impulsketten. Bei Benutzung des CNT-Einganges können die Timer externe Impulse zählen oder auch Frequenzen messen.

Jeder Timer hat ein ihm fest zugeordnetes Steuerregister (CRA und CRB), welches die folgenden Funktionen erlaubt:

Start/Stop (Bit 0)

Dieses Bit läßt den Timer jederzeit starten oder anhalten.

PB ON/OFF (Bit 1)

Hiermit wird der Timer-Unterlauf nach PB geleitet (PB6 für Timer A, PB7 für Timer B). Diese Funktion hat Vorrang vor der in DDRB festgelegten Datenrichtung.

Toggle/Pulse (Bit 2)

Mit diesem Bit wird die Art des an PB erscheinenden Unterlaufpulses bestimmt. Entweder wird PB bei jedem Unterlauf in die jeweils andere Lage gekippt, oder es wird ein positiver Puls mit der Dauer eines Taktes erzeugt.

One-Shot/Continous (Bit 3)

Im One-Shot-Betrieb zählt der Timer vom Zwischenspeicherwert nach Null, setzt das IRC-Bit, lädt den Zähler erneut mit dem Zwischenspeicherwert und hält dann an. Im Continous-Betrieb läuft der oben beschriebene Vorgang zyklisch ab.

Force-Load (Bit 4)

Dieses Bit erlaubt, den Timer jederzeit zu laden, unabhängig davon, ob er gerade läuft oder nicht.

Input-Mode (Bit 5 CRA, Bit 5-6 CRB)

Diese Bits erlauben die Wahl des Taktes, mit dem der Timer heruntergezählt wird. Timer A kann entweder mit dem Systemtakt oder mit einem auf CNT gegebenen Takt versorgt werden. Timer B kann darüber hinaus noch mit den Unterlaufpulsen des Timers A gespeist werden, entweder unbedingt oder in Abhängigkeit von CNT=1.

3.5 Die Echtzeituhr

Im CIA befindet sich eine 24-Stunden-Echtzeituhr *(TOD)* mit einer Auflösung von 1/10 Sekunden. Sie besteht aus den vier Registern: Stunden, Minuten, Sekunden und 1/10-Sekunden. Im Stunden-Register bestimmt das höchstwertige Bit (Bit 7) ob Vormittag *(AM)* oder Nachmittag *(PM)*. Alle Register werden im BCD-Format angegeben, so daß man ohne größeren Rechenaufwand, gerade in Maschinensprache, davon Gebrauch machen kann.

Als Takt dient ein 50/60 Hertz-Signal am Pin TOD, der sich im CRA-Bit 7 programmieren läßt.

Ferner gibt es noch ein Alarmregister, mit dem man zu jeder beliebigen Zeit einen Interrupt auslösen kann. Das Alarmregister belegt dieselbe Adresse wie das TOD-Register, deshalb wird der Zugriff unter Zuhilfenahme des CRB Bit 7 gesteuert. Beachten Sie, daß das Alarmregister Read-Only ist! Jeder Lese-Zugriff gibt das TOD-Register wieder, vollkommen unabhängig von CRB Bit 7.

Um die Uhrzeit richtig setzen und lesen zu können, muß man eine bestimmte Reihenfolge einhalten:

Wird das Stundenregister beschrieben, so hält die Uhr automatisch an - beim Schreiben ins 1/10-Sekunden-Register beginnt die Uhr zu laufen. Auf diese Weise kann man das Starten der Uhr sehr genau steuern.

Da beim Lesen der Uhr ein Übertrag in ein bereits gelesenes Register auftreten kann, werden die Register in einen Zwischenspeicher übertragen. Dieser Zwischenspeicher wird erst dann wieder freigegeben, wenn man die 1/10-Sekunden gelesen hat.

3.5.1 Echtzeit in BASIC

Jeder kennt wahrscheinlich die von BASIC abrufbare Uhr *TI$* und *TI*. Leider läßt die Langzeitgenauigkeit dieser Uhr stark zu wünschen übrig; die Abweichung kann ca. ½ Stunde pro Tag betragen.

Legt man auf eine genauere Zeitangabe Wert, so kann man von der im CIA eingebauten Echtzeituhr durchaus Gebrauch machen. Diese erhält ihren Takt aus der Netzfrequenz, welche eine hervorragende Langzeitkonstanz aufweist.

Hier nun zwei BASIC-Programme, das eine zum Setzen der Uhrzeit und das andere zum Auslesen der Echtzeituhr. Da es unsinnig wäre, die Zehntelsekunden abzufragen, wird dieses Register immer auf Null gesetzt.

```
10 C=56328: REM BASISADRESSE DER UHR IM CIA1
20 REM C=56584 FÜR DIE UHR IM CIA2
30 POKE C+7,PEEK(C+7) AND 127: REM UHRZEIT SETZEN
40 POKE C+6,PEEK(C+6) OR 128 : REM NETZFREQUENZ=50 Hz
50 INPUT "BITTE ZEIT IM FORMAT HHMMSS EINGEBEN: ";A$
60 H=VAL(LEFT$(A$,2))
70 M=VAL(MID$(A$,3,2))
80 S=VAL(MID$(A$,5))
90 IF H>23 THEN 40 : REM FEHLER
100 IF H>11 THEN H=H+68 : REM SETZE NÖTIGENFALLS PM-FLAG
110 POKE C+3,16*INT(H/10)+H-INT(H/10)*10
120 IF M>59 THEN 40 : REM FEHLER
130 POKE C+2,16*INT(M/10)+M-INT(M/10)*10
140 IF S>59 THEN 40 : REM FEHLER
150 POKE C+1,16*INT(S/59)+S-INT(S/59)*10
160 POKE C,0 : REM ZEHNTELSEKUNDEN -- UHR LOS
```

In den Zeilen 110, 130 und 150 werden die Werte in BCD-Format umgewandelt. Um die gesetzte Uhr dann auch wieder auslesen zu können, verwendet man folgendes kleines Programm:

```
 10 C=56328 : REM BASISADRESSE DER UHR IM CIA1
 20 PRINT CHR$(147) : REM C=56328 FÜR UHR IM CIA2
 30 H=PEEK(C+3):M=PEEK(C+2):S=PEEK(C+1):T=PEEK(C)
 40 FL=1
 50 IF H>32 THEN H=H AND 127: FL=0: REM FLAG FÜR PM
 60 H=INT(H/16)*10+H-INT(H/16)*16:ON FL GOTO 80
 70 IF H=12 THEN 90:ELSE H=H+12
 80 IF H=12 THEN H=0
 90 M=INT(M/16)*10+M-INT(M/16)*16
100 S=INT(S/16)*10+S-INT(S/16)*16
110 T$=MID$(STR$(T),2)
120 H$=RIGHT$("0"+MID$(STR$(H),2),2)
130 M$=RIGHT$("0"+MID$(STR$(M),2),2)
140 S$=RIGHT$("0"+MID$(STR$(S),2),2)
150 PRINT "<Home>";
160 PRINT H$;":";M$;":";S$;":";T$
170 GOTO 30 : REM SCHLEIFE
```

Sollten Sie die STOP/RESTORE-Taste drücken, so muß die Uhr allerdings neu gesetzt werden, da das Betriebssystem alle Register auf den Ausgangswert setzt. Leider ist hiervon auch das für den Takt (50/60 Hz) verantwortliche Bit betroffen; die Uhr würde stark zurückbleiben!

3.6 Die CIAs im Commodore 128

Wenn Sie von den CIAs im Commodore 128 Gebrauch machen wollen, so beachten Sie bitte, daß die CIAs fest zugeordnete Aufgaben zu erfüllen haben. Insbesonders gilt das für die Interrupts, die das Betriebssystem für eine Reihe von Routinen benötigt. Ändern Sie also möglichst nie das Register ICR.

CIA 1: Basisadresse $DC00 (56320)

REG 0 (PRA)
- Bits 0-7: Im normalen Betrieb wird hier die Reihenauswahl der Tastaturmatrix getroffen. Allerdings sind einige Bits mit dem Controllerport 1 außen am Rechner verbunden. Dieser dient zum Anschluß von Joysticks oder Paddles,
- Bits 0-4: Joystick 0, Reihenfolge: Oben, Unten, (links, rechts und Feuertaste),
- Bits 6-7: Auswahl Paddle-Set A/B. Es darf nur eines der beiden Bits = 1 sein.

REG 1 (PRB)
- Bits 0-7: Im normalen Betrieb erfolgt hier die Spaltenrückmeldung der Tastaturmatrix, falls eine Taste gedrückt wurde,
- Bits 0-4: Dieselbe Funktion wie REG 0, allerdings für Controlport 2 (Joystick 1).

REG 13 (ICR)
- Bit 4: Eingabedaten vom Kassettenport.

Timer A & CRA werden für den Diskettenbetrieb benötigt, Timer B & CRB für den Kassettenbetrieb.

CIA 2: Basisadresse $DD00 (56576)

REG 0 (PRA)
- Bits 0-1: VA 14-15 (höchstwertige Adreßbits des Video-RAM),
- Bit 2: TXD (in Verbindung mit RS232-Cartridge, sonst frei),
- Bit 3: ATN (Ausgang serieller Bus),
- Bit 4: CLOCK (Ausgang serieller Bus),
- Bit 5: DATA (Ausgang serieller Bus),
- Bit 6: CLOCK (Eingang serieller Bus),
- Bit 7: DATA (Eingang serieller Bus).

REG 1 (PRB)
- Bits 0-7: User-Port/RS232. Bei Aufstecken einer RS232-Cartridge erhalten die Bits die folgende Bedeutung:
- Bit 0: RXD (Receive Data),
- Bit 1: RTS (Request To Send),
- Bit 2: DTR (Data Terminal Ready),
- Bit 3: RI (Ring Indicator),
- Bit 4: DCD (Data Carrier Detect),
- Bit 6: CTS (Clear To Send),
- Bit 7: DSR (Data Set Ready).

REG 13 (ICR)
Bit 4: RXD (nur bei RS232-Betrieb, sonst frei).

Timer A & CRA werden für die RS232-Baudrate benötigt, Timer B & CRB für die RS232-Bitüberprüfung.

3.7 Der Joystickport

3.7.1 Der Joystick

Wollen Sie nicht die in BASIC 7.0 implementierten Kommandos zur Joystickabfrage verwenden, oder wollen Sie eine Joystickabfrage in Maschinensprache realisieren, so können Sie folgendes kleines BASIC-Programm zur Interpretation der anfallenden Daten zur Hilfe nehmen:

```
10 J0=56320 : REM JOYSTICK-PORT-1
20 J1=56321 : REM JOYSTICK-PORT-2
30 J=PEEK(J0) : REM AUSLESEN DES PORTS
40 IF (J AND 1)=0 THEN PRINT "OBEN ";
50 IF (J AND 2)=0 THEN PRINT "UNTEN ";
60 IF (J AND 4)=0 THEN PRINT "LINKS ";
70 IF (J AND 8)=0 THEN PRINT "RECHTS ";
80 IF (J AND 16)=0 THEN PRINT "FEUER";
90 PRINT: GOTO 30
```

Das Programm liest nun den Joystick-Port-1 aus; wollen Sie Port 2 auslesen, so brauchen Sie lediglich in Zeile 30 *J0* durch *J1* zu ersetzen.

Sollten Sie in zwei Richtungen gleichzeitig steuern wollen, beispielsweise nach oben und rechts, so ist auch diese Abfrage möglich - in unserem Beispiel werden Ihnen beide Richtungen auf dem Bildschirm angezeigt. Auf diese Weise erhöht sich die Anzahl der Richtungen von 4 auf 8! Lesen Sie aber auch die Verwendung von Paddles in Kapitel 4.1.4.2.

3.7.2 Die 1350-Maus

Auch auf die mittlerweile mehr oder weniger modern gewordene Maus braucht der Commodore-128-Besitzer nicht mehr länger zu verzichten. Mit der Maus können Sie Programmfunktionen auswählen und ausführen ohne von der Tastatur Gebrauch machen zu müssen, sofern dies vom Treiberprogramm unterstützt wird.

Genauso wie der Joystick, der Light Pen oder die Paddles wird die Maus an den Joystickport angeschlossen. Dabei hat die linke Maustaste dieselbe Auswirkung wie die Feuertaste des Joysticks; die rechte Taste hat softwarebedingte Zusatzfunktionen.

Jetzt das angenehmste: Die Maus kann genauso abgefragt werden wie ein Joystick. So funktionieren alle Programme mit Joystickabfrage auch mit der Maus; selbst im 64er-Modus! Unter BASIC 7.0 können Sie hier von dem JOY-Kommando Gebrauch machen. Benutzen Sie doch aber einfach mal das hier abgedruckte Joystickprogramm.

3.7.3 Der Light Pen

Auch den Light Pen schließt man am Joystickport an. Mittels des Light Pen kann man softwaremäßig feststellen, auf welche Stelle im Bildschirm der Stift zeigt. Hierfür wird der VIC-Chip benötigt, der die nötigen Informationen der aktuellen Kathodenstrahlposition mitteilt. Die Position des Light Pen wird an Adresse $D013 (X-Position) übergeben. Da die X-Koordinate des Kathodenstrahl bis 512 betragen kann (9 Bits), ist die Auflösung des Light Pen auf 256 Punkte herunterdividiert worden (2 horizontale Punkte = 1 Light-Pen-Punkt). Entsprechend finden Sie die Y-Koordinate des Light Pen an $D014 vor, mit voller Auflösung übrigens.

Die Auflösung des Light Pen ist von Modell zu Modell verschieden und auch vom Monitor abhängig. Deswegen verlangen die meisten Programme, die mit Light Pen arbeiten, daß man den Light Pen erst einmal herumbewegt, das Programm stellt sich dann auf den Light Pen ein. BASIC-Programmierer

haben es bei der Light-Pen-Abfrage sehr leicht, sie können das PEN-Kommando verwenden.

```
10 REM ** Light-Pen-Abfrage ***
20 GRAPHIC 1,1
30 COLOR 4,2 : COLOR 0,2 : COLOR 1,0
40 DO
50 :    X = PEN(0)
60 :    Y = PEN(0)
70 :    DRAW 1, X, Y
80 LOOP
```

3.8 Der serielle IEC-Bus des Commodore 128

Am IEC-Bus werden Peripherie-Geräte an den Rechner angeschlossen. Dies können beispielsweise Drucker oder Floppy-Laufwerke sein. Sie dürfen sich den *Bus* tatsächlich als einen solchen vorstellen: Es werden Daten vom Rechner über den Bus an bestimmte Haltestellen (Peripherie) transportiert, und sie kommen auf demselben Weg auch wieder zurück. Der im Commodore 64 und Commodore 128 eingebaute IEC-Bus ist eine abgespeckte Version der in den "größeren" CBM-Anlagen eingebauten IEC-Busse. Der große IEC-Bus hat 24 Leitungen, der kleinere Bus hat lediglich 6 Leitungen. Diese Verkleinerung mag aus Kosten- oder Platzgründen geschehen sein, auf jeden Fall trug dieser IEC-Bus für Rechner (es gibt auch einen IEC-Bus für Meßgeräte) nicht unwesentlich zum großen Erfolg der Commodore-Rechner bei. (Manche behaupten sogar, er sei das Geheimrezept von Commodore).

Doch hier erst einmal die Belegung der einzelnen Leitungen:

1 SRQ; Service ReQuest. Hat ein Gerät irgendeine Aufgabe erledigt und braucht nun neue Daten oder hat welche abzugeben oder verlangt sonst irgendeine Aktion, so kann es das dem Controller durch dieses Signal mitteilen. (Wie im Krankenhaus, wo Sie nach der Schwester klingeln können.) Dieser wird daraufhin einen Identify-Zyklus (mittels EOI oder ATN) einleiten, um festzustellen, um welches Gerät es sich handelt. Diese Funktion wird bei Commodore nicht verwendet.

2 GND; Masseleitung.

3 ATN; (In) ATteNtion. Immer dann, wenn der Controller einen Befehl übermitteln will, aktiviert er diese Leitung. Dadurch soll erreicht werden, daß alle am Bus angeschlossenen Geräte diesen Befehl mitbekommen (sie sollen "lauschen"), da ja von vorneherein noch nicht feststeht, welches Gerät gemeint ist. Dies stellt sich erst bei der Übermittlung der Geräteadresse heraus, weshalb diese auch immer zuerst übermittelt wird, damit sich die anderen Geräte wieder vom Bus trennen können.

4 CLK; (In/Out) CLocK. Da die Daten nicht byteweise, sondern bitseriell über den Bus wandern, gibt der TALKER jedem Bit einen Taktimpuls auf der Leitung CLK mit auf dem Weg, womit die Gültigkeit der Datenleitung angezeigt wird.

5 DATA (In/Out) ist die einzige Datenleitung, über die ein Datenbyte mit dem niederwertigsten Bit voran seriell geschoben wird.

6 RESET; Gibt einen Reset auf die angeschlossenen Geräte.



Abb. 3

Alle auf dem größeren IEC-Bus vorhandenen zusätzlichen Leitungen, wie etwa EOI, NDAC etc. werden durch die beiden Leitungen CLK und DATA simuliert oder besser ersetzt. Die Zeit zwischen den Pegelsprüngen der beiden Leitungen gibt Auskunft über das jeweilige Signal.

3.8.1 Schneller und langsamer Modus

Sicherlich meinen Sie auch, daß es eine sträfliche Verschwendung ist, eine Leitung auf dem ohnehin mickrigen Bus unbenutzt zu lassen. Doch leider ist dies so - zumindest im *normalen* Modus.

Wenn es einen normalen Modus gibt, so schließen Sie, dann gibt es auch einen anderen, nicht normalen Modus. Dies ist genau richtig! Wie Sie wissen, ist die 1541 nicht gerade als schnelle Floppy verschrien (ganz im Gegenteil). Das liegt schon allein daran, daß jedes Byte zerpflückt werden muß und dann Bit für Bit über den Bus geschickt wird. Dieser Mißstand mußte natürlich behoben werden - was nutzt einem die Supermaschine Commodore 128, wenn man einen solchen Klotz am Bein hat? Schließlich und endlich schläft die Konkurrenz auch nicht, sprich die Konkurrenzprodukte werden auch nicht langsamer. Commodore hat hier die Floppy 1571 entwickelt, die beim Ladevorgang im 128er-Modus bis zu achtmal (!) schneller ist. Im CP/M-Modus wird dann sogar noch einiges zugelegt. Diesen Geschwindigkeitsvorteil hat man allerdings *nur* im 128er-Modus, nicht im 64er-Modus. Die 1541 kann man allerdings nach wie vor im 128er-Modus betreiben. Die Floppy erkennt soweit selbständig, welcher Modus eingeschaltet werden muß; dies aber nur in Kooperation mit dem Betriebssystem des Rechners.

Sie können sich bestimmt schon denken, wie man diesen Geschwindigkeitsvorteil realisiert hat: unter Zuhilfenahme der brachliegenden Leitung SRQ. Im schnellen seriellen Modus wird diese Leitung praktisch als eine zweite CLK-Leitung benutzt, als bidirektionale schnelle CLOCK-Leitung.

Im Einschaltzustand ist die 1571 immer im langsameren Modus, weshalb man sie auch ohne weiteres am C-64 anschließen kann. Der Benutzer kann nun den *schnellen* Modus anmelden, der dann beibehalten wird, bis er wieder explizit aufgehoben wird. Die existierenden Kernal-Routinen sind geändert worden, um den langsamen und schnellen Modus zu realisieren. Es gibt ein spezielles Flag im Kernal, um anzuzeigen, ob das aktuelle Peripheriegerät ein schnelles oder ein langsames ist.

Um die 1571 als schnelles Gerät anzumelden, muß der Benutzer ein HRF-Signal senden (*Host Request Fast* = Host erbittet schnellen Modus). Dies wird durch Senden von acht CLOCK-Impulsen über die SRQ-Leitung realisiert. Der 6526 auf der Controller-Platine des Laufwerkes erkennt dieses Signal und löst einen Interrupt aus. Im Laufwerk wird dann ein Flag gesetzt, das den schnellen Modus anzeigt. Wenn die Floppy nun *LISTENER* ist und Daten empfängt, sendet sie das ein DRF-Signal (*Device Request Fast* = Gerät erbittet schnellen Modus). Anhand dieses Signals erkennt dann der Rechner, daß die Floppy in der Lage ist, Daten schnell zu empfangen und zu senden; eine 1541 würde dieses DRF-Signal natürlich nicht senden können. Das rechnerinterne Schnell-Modus-Flag kann durch folgende Ereignisse wieder rückgesetzt werden: UNLISTEN, UNTALK, Fehler auf Bus sowie <RUN/STOP>-<RESTORE>.

3.8.2 Die Geräteadressen

Es besteht die Möglichkeit, eine Reihe von Geräten an den seriellen IEC-Bus anzuschließen, bespielsweise zwei Floppies und einen Drucker. Dies macht es notwendig, die verschiedenen Geräte auch unterscheiden zu können, damit die Daten wissen, wo sie "aussteigen" müssen. Sie können sich die Geräteadresse also praktisch als Hausnummer vorstellen. Als Geräteadresse sind die Werte 0-30 möglich.

Geräteadresse	Gerät
0-3	Interne Geräte (Tastatur, Bildschirm, User Port, Kassettenport).
4-7	Normalerweise CBM-Drucker.
8-11	Normalerweise CBM-Floppies.
12-30	Noch unbenutzt.

Die Geräteadresse enthält allerdings außer der eigentlichen Gerätenummer noch eine weitere Information: die Aktion, die ausgeführt werden soll. Die möglichen Aktionen sind die folgenden:

32 Das Gerät wird als LISTENER adressiert, d.h. es soll Daten empfangen. Diese Aktion wird beispielsweise durch das BASIC-Kommando PRINT# oder DSAVE hervorgerufen.

64 Das Gerät soll TALKER sein, also Daten senden. Hier könnten beispielsweise die BASIC-Kommandos INPUT# oder DLOAD als Ursache dienen.

48 Die Betriebsart LISTEN wird beendet (UNLISTEN). Das niederwertige Halbbyte (Gerät) ist hierbei immer 15.

80 Die Betriebsart TALK wird beendet (UNTALK). Das niederwertige Byte ist auch hier immer 15.

Wollen Sie beispielsweise einen Drucker mit der Geräteadresse 4 zum Drucken adressieren, so setzt sich die gesamte Geräteadresse zusammen aus: 32 + 4 = 36 ($24).

3.8.3 Die Sekundäradressen

Die Sekundäradresse dient nicht der Auswahl eines Gerätes am IEC-Bus - sie dient vielmehr der Auswahl eines Modus im angesprochenen Gerät. So kann man beispielsweise bei den meisten Druckern durch Angabe der Sekundäradresse einen ganz bestimmten Druckmodus wählen. Bei den CBM-Druckern kann man beispielsweise durch die Sekundäradresse 0 den Groß-/Grafik-Modus anwählen, die Sekundäradresse 7 wählt den Groß-/Kleinschrift-Modus aus. Auch beim Floppy-Laufwerk kann man durch Angabe der Sekundäradresse einen Datenkanal ansprechen.

Auch die Sekundäradresse setzt sich zusammen aus der eigentlichen Sekundäradresse und dem Zusammenhang, in dem die Sekundäradresse auftritt.

96 PRINT, INPUT oder GET
224 CLOSE
240 OPEN

Als nützlich wird sich auch diese Tabelle erweisen, die Ihnen in Bitmustern anzeigt, wie sich die einzelnen Geräte- und Sekundäradressen zusammensetzen.

Kommando	Abkürzung	Binärer Wert
Host Request Fast	HRF	%1111 1111
Device Request Fast	DRF	%0000 0000
Talk Adresse	(TA)	%010x xxxx
Listen Adresse	(LA)	%001x xxxx
UNTALK	(UNTLK)	%0101 1111
UNLISTEN	(UNLSN)	%0011 1111
SA OPEN	(SA(O))	%1111 yyyy
SA CLOSE	(SA(C))	%1110 yyyy
SA Normal	(SA)	%011z zzzz

Die normale Sekundäradresse darf einen Wert zwischen 0 und 31 haben (z zzzz). Die Kanaladresse (yyyy) darf einen Wert zwischen 0 und 15 haben. Als Beispiel die Sekundäradressen und deren Bedeutung für die 154x-Laufwerke:

```
   00 - PRG-Type (Read Data Channel)
   01 - PRG-Type (Write Data Channel)
02-14 - Kanäle für alle Filetypen
   15 - Kommandokanal
```

3.8.4 Die Systemvariable ST

Wo Peripheriegeräte angeschlossen sind, können natürlich auch Fehler auftreten. Die Systemvariable *ST* gibt Auskunft darüber, ob die letzte Aktion auf dem IEC-Bus erfolgreich verlaufen ist oder nicht; ist sie nicht erfolgreich verlaufen, so kann man den Fehler anhand des in ST übergebenen Fehlercodes analysieren. ST kann nun folgende Werte annehmen:

1 Kann nach OPEN oder PRINT auftreten. Nach der Übergabe eines Datenbytes wurde dieses nicht innerhalb von 64 ms durch Zurücknahme von NDAC quittiert, es ist also wahrscheinlich nicht angekommen.

2 Kann bei INPUT oder GET auftreten. Wird ein Gerät als TALKER angesprochen und sendet nicht innerhalb von 64 ms ein Datenbyte, so beinhaltet ST diesen Wert.

64 Das zuletzt übergebene Datenbyte wurde in Verbindung mit einem EOI übergeben, was bei einer Floppy auf Dateiende hindeutet (EOF).

-128 Ein Adressierungsversuch zeigte keinerlei Reaktion auf dem Bus. In diesem Falle wird ein BASIC-Programm die Fehlermeldung DEVICE NOT PRESENT ausgeben, in Maschinensprache können Sie hierauf reagieren.

Es kann auch vorkommen, daß eine Kombination der oben genannten Werte auftritt. Hier ist es ratsam, in einem BASIC-Programm nicht den absoluten Wert abzufragen, sondern das entsprechende Bit, also:

```
1000 IF (ST AND 64) THEN PRINT "<EOF>"
```

Um in Maschinensprache das Statuswort ST abzufragen, ist es notwendig, sich dieses aus der Zeropage zu holen. Erfreulicherweise liegt es sowohl im 64er- als auch im 128er-Modus an derselben Adresse: $90 (144). Ein Abfragen in Maschinensprache sähe dann so aus:

```
LDA $90   ;Hole Statusvariable
AND #$40  ;Bit 6 gesetzt?
BNE EOF   ;EOF erreicht
   .
   .
```

4. Der Soundchip SID

4.1 Der Soundcontroller 6581

4.1.1 Allgmeines zum SID

Auch die Musik ist - neben der Grafik - ein interessantes Anwendungsgebiet in der Computerei. Sie können sich glücklich schätzen, da ein überaus leistungsstarker Synthesizer - etwas anderes ist der SID nicht - in Ihrem Commodore schlummert. Es handelt sich um denselben Baustein, wie er sich schon im Commodore 64 bewährt hat. Kaum ein Spiel verzichtet auf die Möglichkeiten, die sich durch den SID bieten, aber keines erreicht wohl die Leistungsgrenzen dieses Chips. Nicht selten ertönen die bekanntesten Melodien in allen möglichen und unmöglichen Klangfarben - Sprechen kann er mittlerweile auch, der Commodore, natürlich dank des SIDs, ohne jeglichen weiteren Hardware-Aufwand. Benötigt wird lediglich ein entsprechendes Programm.

Hier die wesentlichen Merkmale des SID 6581 in Stichpunkten:

* *3 unabhängige, frei programmierbare Stimmen*
* *4 mischbare Schwingungsarten für jede Stimme*
* *3 mischbare Filter (Hoch-, Tief-, Bandpassfilter)*
* *Hüllkurvengenerator (ASDR-Kontrolle) für jede Stimme*
* *2 kaskadierbare Ringmodulatoren*
* *Verfremdungsmöglichkeit für externe Signalquelle*
* *Zwei 8-Bit-A/D-Wandler*

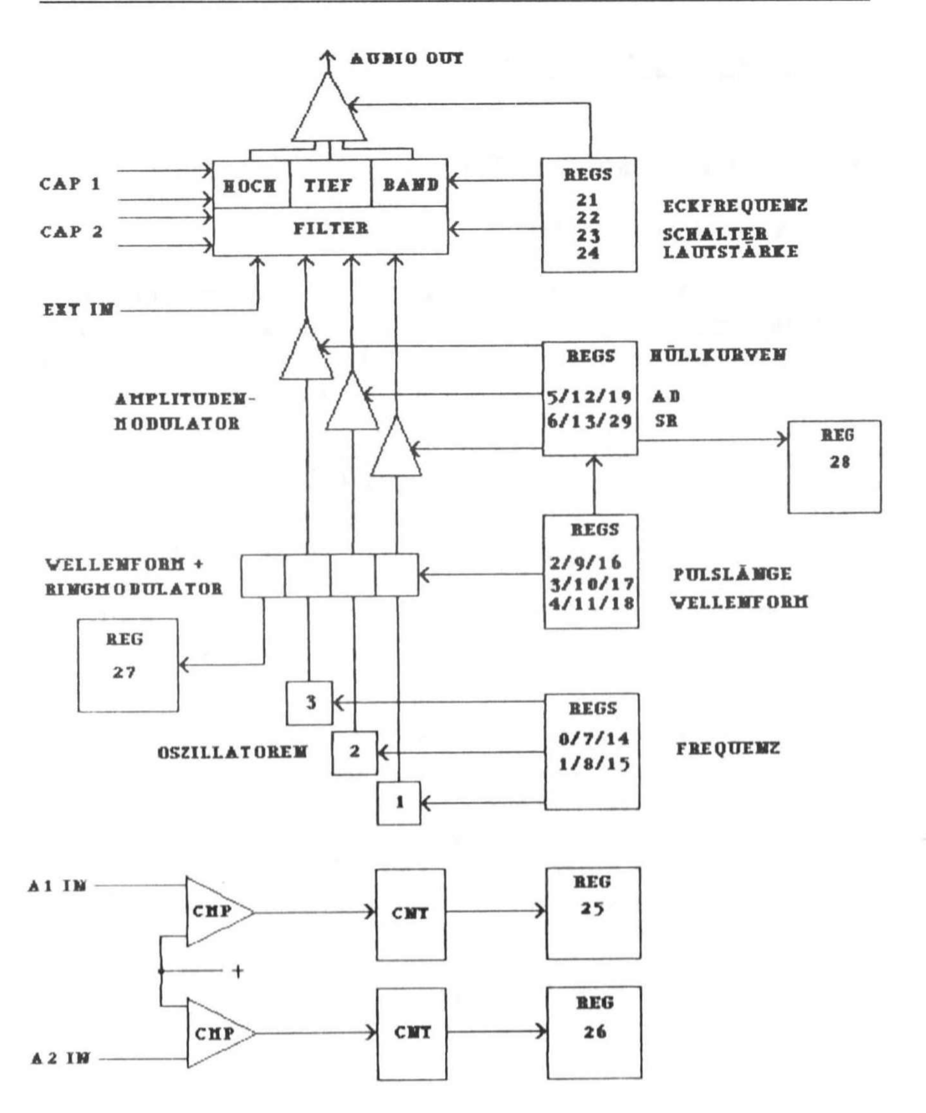


Abb. 4 (Blockschaltbild SID)

4.1.2 Pinbelegung des 28 poligen Gehäuses:

1-2	CAP1A, CAP1B; Anschluß des Kondensators für die programmierbaren Filter. Empfohlene Kapazität: 2200pF.
3-4	CAP2A, CAP2B; wie 1-2.
5	-RES (Reset); =0 bringt den SID in den Grundzustand.
6	02 (Systemtakt); alle Datenbusaktionen finden nur während 02=1 statt.
7	R/W (Read/Write); 0=Schreibzugriff, 1=Lesezugriff.
8	-CS (Chip Select); 0=Datenbus gültig, 1=Datenbus hochohmig (Tri-State).
9-13	A0-A4 (Adreßbits 0-4); dienen zur Auswahl eines der 29 Register des SID.
14	GND (Masse); Achtung: Der SID sollte eine eigene Masseleitung zur Stromversorgung besitzen, um gegenseitige Beeinflussungen mit anderen Systemkomponenten zu vermeiden!
15-22	D0-D7; Datenleitungen von und zum Prozessorsystem.
23	A2IN (Analog Input 2); Handhabung unbedingt unter 4.1.4 nachlesen!
24	A1IN; Wie 23, allerdings für A/D-Wandler 1.
25	VCC; Versorgungsspannung +5V.
26	EXT IN (External Input); Eingang für externe Audiosignale, die durch SID verfremdet werden sollen.
27	AUDIO OUT; Summenausgang aller im SID erzeugten oder behandelten Signale.
28	VDD; Versorgungsspannung +12V.

SID steht für *Sound Interface Device*. Während handelsübliche Synthesizer meist nur über eine Stimme verfügen, man nennt sie dann *monophon*, so verfügt der SID gleich über drei voneinander unabhängige, frei programmierbare Stimmen (*polyphon*). Viele Konkurenzrechner haben sich auf dieses leistungsstarke Element eingestellt und ebenfalls mehrstimmige Synthesizer eingebaut.

Wie bereits erwähnt, verfügt der SID 6581 über drei frei programmierbare Stimmen, die sowohl unabhängig voneinander als auch in Verbindung miteinander benutzt werden können.

Einige der Leser haben mit Sicherheit schon Töne oder Tonfolgen unter BASIC 7.0 programmiert. Auch bei der Produktion von Tönen müssen Sie sich von diesem Standard lösen, der Ihnen durch BASIC 7.0 geboten wird: Der SID ist zwar komfortabel, aber Dinge wie Warteschlangen (Queues) etc. müssen Sie in Maschinensprache selbst durch Interrupts lösen. Im 64er-Modus stehen Ihnen die komfortableren Kommandos auch nicht zur Verfügung - was aber kein Grund zur Resignation ist, denn man kann auch durch entsprechende POKE-Kommandos eine Menge

aus dem SID herausholen; im Prinzip macht der Interpreter bei der Ausführung Ihrer BASIC-Kommandos ja nichts anderes.

Wer also schon einmal unter BASIC Töne programmiert hat oder wer Synthesizer-Freak ist, für den sind Begriffe wie *Hüllkurve* und *Amplitudenmodulator* nichts Unbekanntes mehr. Wir wollen diese so notwendigen Begriffe aber auf jeden Fall erläutern, da sie zur Programmierung des SID sehr wichtig sind.

Jede Stimme besteht aus: *Oszillator*, *Hüllkurvengenerator*, *Amplitudenmodulator* und einem *Schwingungsformgenerator*. Der Oszillator erzeugt bei einer Taktfrequenz von 1 MHz eine Grundfrequenz im Bereich 0 - 8200 Hz mit einer Auflösung von 16 Bit. Es sind vier verschiedene Schwingungsformen möglich: Sägezahn, Rechteck (mit variablem Puls-/Pauseverhältnis), Dreieck und das jedem eingefleischten HiFi-Freak bekannte "rosa Rauschen". Die Schwingungsform ist ein wichtiges Kriterium für das Klangbild des zu erzeugenden Tones, da jede Schwingungsform ihren eigentümlichen Gehalt an Oberwellen hat. So hört sich die Dreiecksschwingung sehr weich an, etwa wie eine Holzflöte. Die Sägezahnschwingung klingt eher blechern, einer Trompete sehr ähnlich. Einer Klarinette gleicht die Rechteckschwingung, die sehr hohl klingt. Bleibt noch das Rauschen übrig, das weniger einem Instrument ähnelt, obwohl man ein Schlagzeug recht gut simulieren kann. Geräuscheffekte beispielsweise können am besten durch das Rauschen, eventuell durch Überlagerung einer weiteren Welle, erzeugt werden. Das Rauschen wird durch Überlagerung einer Vielzahl aufeinanderfolgender, zufälliger Frequenzen erreicht.

Der Amplitudenmodulator beeinflußt den Verlauf der Lautstärke während der Erzeugung eines Tones. Der Modulator wird durch den Hüllkurvengenerator gesteuert, den Sie wiederum direkt programmieren können. Um die Programmierung des Hüllkurvengenerators kümmern wir uns später noch. Ferner können die Ausgänge aller Stimmen weiterhin noch über programmierbare Filter geschickt werden, wodurch Sie die Klangfarbe noch einmal beeinflussen können. Für SID-Fans gibt es dann noch einige Möglichkeiten: Es können die Stimmen 1 und 2 noch von der Stimme 3 ringmoduliert werden, d.h. es entstehen außer der Grundstimme noch Summe und Differenz mit der Stimme 3. Bei der Stimme 3 kann man während des

Verlaufes eines Tones den Augenblickswert des Hüllkurvengenerators auslesen und dann anhand dieser Daten beispielsweise einen Filter verändern.

4.1.3 Registerbeschreibung des SID

Die Basisadresse des SID 6581 ist $D400 (54272).

REG 0 Oszillatorfrequenz niederwertiges Byte für Stimme 1.

REG 1 Oszillatorfrequenz höherwertiges Byte für Stimme 1.

REG 2 Pulsbreits niederwertiges Byte für Stimme 1.

REG 3 Pulsbreite höherwertiges Byte für Stimme 1.
Die Register 2 und 3 bestimmen das Puls-/Pauseverhältnis des Rechteckausganges von Stimme 1. Von Register 3 werden lediglich die Bits 0-3 benutzt.

REG 4 Steuerregister für Stimme 1

- Bit 0: KEY; Steuerbit für den Ablauf des Hüllkurvengenerators. Beim Wechsel von 0 nach 1 steigt die Lautstärke von Stimme 1 innerhalb der in REG 5 programmierten "Attack"-Zeit von Null auf den Maximalwert (REG 24) an und fällt dann in der ebenfalls in REG 5 angegebenen "Decay"-Zeit auf den in REG 6 programmierten "Sustain"-Pegel ab, auf dem sie bleibt, bis das Steuerbit wieder Null wird. Dann fällt die Lautstärke innerhalb der in REG 6 ausgewählten "Release"-Zeit wieder auf Null ab.
- Bit 1: SYNC; 1= Oszillator 1 wird mit Oszillator 3 synchronisiert. Dieses Bit hat auch dann Wirkung, wenn die Stimme 3 stummgeschaltet sein sollte.
- Bit 2: RING; 1= Der Dreieckschwingungsausgang von Oszillator 1 wird durch ein Frequenzgemisch (Summe und Differenz der Frequenzen von Stimme 1 und 3) ersetzt. Dieser Effekt tritt auch dann ein, wenn Stimme 3 stummgeschaltet ist.
- Bit 3: TEST; Wenn zusammen mit dem Rauschgenerator noch eine weitere Schwingungsform desselben Oszillators ausgewählt wurde, kann es vorkommen,daß der Rauschgenerator blockiert. Die Blockade kann durch dieses Bit wieder aufgehoben werden.
- Bit 4: TRI; 1= Dreieckschwingung ausgewählt.
- Bit 5: SAW; 1= Sägezahnschwingung ausgewählt.
- Bit 6: PUL; 1= Rechteckschwingung ausgewählt. Das Puls-/Pauseverhältnis dieser Schwingung wird in REG 2 und REG 3 eingestellt.
- Bit 7: NSE; 1= Rauschgenerator ausgewählt.

Anmerkung zu den Bits 4-7: Es ist praktisch möglich, mehrere Schwingungsformen gleichzeitig auszuwählen. Zu beachten ist jedoch, außer dem zu Bit 3 Gesagten, daß das Ergebnis nicht etwa die Summe aller Formen darstellt sondern vielmehr eine logische UND-Verknüpfung der Komponenten.

REG 5 ATTACK/DECAY

Bits 0-3: Diese Bits bestimmen die Zeit, die verstreicht, bis die Lautstärke vom Maximalwert auf den Sustain-Pegel abfällt. Der einstellbare Bereich beträgt 6 ms bis 24 Sekunden.

Bits 4-7: Hiermit wird die Zeit definiert, in der die Lautstärke nach dem Setzen des KEY-Bits von Null auf Maximalwert ansteigt. Der einstellbare Bereich liegt hier bei 2 ms bis 8 Sekunden.

REG 6 SUSTAIN/RELEASE

Bits 0-3: Mit diesen Bits wird die Zeit eingestellt, innerhalb der die Lautstärke nach Rücksetzen des KEY-Bits (Ende des Tones) vom Sustain-Pegel auf Null abfällt. Der einstellbare Bereich ist 6 ms bis 24 Sekunden.

Bits 4-7: Diese Bits geben den Sustain-Pegel an, also die Lautstärke, die nach dem Ansteigen auf Maximalwert und folgendem Abfallen gehalten werden soll.

REG 7 bis REG 13 Diese Register steuern die Stimme 2 analog zu den Registern 0-6, allerdings mit folgenden Ausnahmen:
SYNC synchronisiert Oszillator 2 mit Oszillator 3.
RING ersetzt den Dreieckausgang von Oszillator 3 durch das Frequenzgemisch aus den Oszillatoren 2 und 3.

REG 14 bis REG 20 Diese Register steuern die Stimme 3 analog zu den Registern 0-6 mit folgenden Ausnahmen:
SYNC synchronisiert Oszillator 3 mit Oszillator 2.
RING ersetzt die Dreieckschwingung von Oszillator 3 durch das Frequenzgemisch aus den Oszillatoren 2 und 3.

REG 21 Filterfrequenz niederwertiges Byte
Es werden nur die Bits 0-2 benutzt.

REG 22 Filterfrequenz höherwertiges Byte
Die 11-Bit-Zahl der Register 21 und 22 bestimmt die Filtereckfrequenz bzw. -mittenfrequenz.
Im Commodore 128 errechnet sich diese Frequenz folgendermaßen:
F=(30+W*5.8) Hz, wobei W die 11-Bit-Zahl darstellt.

REG 23 Filterresonanz und -schalter

Bit 0:	1= Stimme 1 wird über den Filter geleitet.
Bit 1:	1= Stimme 2 wird über den Filter geleitet.
Bit 2:	1= Stimme 3 wird über den Filter geleitet.
Bit 3:	1= Die externe Quelle wird über den Filter geleitet.
Bits 4-7:	Diese Bits bestimmen die Resonanzfrequenz des Filters. Diese benutzt man dazu, bestimmte Ausschnitte des Frequenzspektrums hervorzuheben. Die Wirkung kann besonders gut bei der Sägezahnschwingung beobachtet werden.

REG 24 Dieses Register hat folgende Bedeutungen:

Bits 0-3:	Gesamtlautstärke.
Bit 4:	Schaltet den Tiefpaßzweig des Filters ein.
Bit 5:	Schaltet den Bandpaßzweig des Filters ein.
Bit 6:	Schaltet den Hochpaßzweig des Filters ein.

Hoch- und Tiefpaßfilter haben eine Flankensteilheit von 12 db/Oktave. Der Bandpaßfilter hat eine solche von 6 db/Oktave. Es kann mehr als ein Filter eingeschaltet sein. Sind beispielsweise Hoch- und Tiefpaß eingeschaltet, resultiert daraus eine Bandsperre. Um den Einfluß der Filter zu Gehör zu bringen, muß wenigstens ein Filter eingeschaltet sein und wenigstens eine Stimme über den Filter geleitet werden. Allgemein gesehen wird das Filter dazu benutzt, bestimmte Bereiche aus einem Frequenzspektrum herauszufiltern. Daher erlaubt die Filterung eine viel feinfühligere und ausgeklügeltere Beeinflussung des Klangbildes, als es durch die bloße Auswahl der Schwingungsform möglich wäre.

Verändert man die Filterfrequenz während des Ablaufes eines Tones (bei kurzen Tönen nur in Maschinensprache möglich), so lassen sich die verschiedensten Instrumente perfekt nachbilden.

Bit 7:	1=Stimme 3 unhörbar. Von dieser Möglichkeit sollte man dann Gebrauch machen, wenn der Verlauf der Stimme 3 lediglich zur Parametergewinnung der anderen Stimmen "mißbraucht" wird.

Auf alle zuvor aufgeführten Register kann nur ein Schreibzugriff erfolgen. Ein Lesezugriff bringt keinerlei Aussage. Auf alle folgenden Register kann nur lesend zugegriffen werden:

REG 25 A/D-Wandler 1

REG 26 A/D-Wandler 2

REG 27 Rauschgenerator der Stimme 3

Dieses Register liefert eine Zufallszahl, die dem augenblicklichen Stand des Rauschgenerators 3 entspricht. Der Generator muß hierzu eingeschaltet sein, jedoch kann die Stimme 3 unhörbar sein (Bit 7 in REG 24 =1).

REG 28 Hüllkurvengenerator der Stimme 3

Aus diesem Register kann man den augenblicklichen Stand der relativen Lautstärke von Stimme 3 entnehmen. So könnte man entsprechend dem Lautstärkeverlauf die Frequenz oder die Filterparameter variieren. Ein Beispiel hierzu finden Sie im Abschnitt 4.2.2.

Nachdem wir eine Tabelle der Register haben, wollen wir anhand kurzer Beispiele auch die Verwendung dieser Register erläutern. Wir setzen den Schwerpunkt allerdings auf die tonerzeugenden Register, nicht auf die A/D-Wandler etc.

4.1.4 Der Analog/Digitalwandler

Die Worte *analog* und *digital* dürften weitläufig bekannt sein, beispielsweise nennt man bei Quarzuhren die Uhren mit den guten alten Zeigern analog, während die Uhren mit LCD-Anzeige digital genannt werden. Diese Bezeichnungen rühren von der Art und Weise her, wie die Zeit angezeigt wird.

Ein A/D-Wandler (so nennt man ihn kurz) ist eine Einrichtung zur Umwandlung eines analogen Signales, beispielsweise einer Spannung, in einen digitalen Wert. Das Problem ist, daß man einen analogen Wert mit theoretisch unendlich feiner Abstufung in einen endlich, durch feste Intervalle abgestuften digitalen Wert umwandeln muß. Es entsteht bei dieser Umwandlung ein maximaler Fehler von ± einem kleinsten digitalen Schritt.

Wie Sie den Registern entnehmen können, enthält der SID 6581 zwei A/D-Wandler. Hierbei handelt es sich um eine Anordnung mit einer intern erzeugten Referenzspannung von ca. 2.5 V.

Der Meßvorgang besteht darin, daß eine externe Kapazität zunächst entladen wird und dann anschließend ein Wert in Register 25 bzw. Register 26 übernommen wird, der der benötigten Zeit für eine erneute Aufladung der Kapazität auf die Referenzspannung entspricht. Dieser Vorgang wiederholt sich zyklisch.

4.1.4.1 Die Handhabung des A/D-Wandlers

Wir schließen aus dem Erwähnten, daß nur eine potentiometrische Beschaltung des Wandlers in Frage kommt. Als Meßwertaufnehmer eignen sich dann dementsprechend nur veränderliche Widerstände in irgendeiner Form, beispielsweise Photowiderstände, Heiß- und Kaltleiter etc.

Sollen Spannungen gemessen werden, so müssen diese zuvor in eine geeignete Form umgewandelt werden, z.B. mit Hilfe eines Unijunction-Transistors.

Die Meßanordnung sieht einfach so aus, daß an das eine Ende des Meßwiderstandes +5V angelegt werden (an den Controlports des C128 verfügbar) und das andere Ende mit dem Analogeingang des SID (ebenfalls an den Controlports verfügbar, die Bezeichnungen lauten *POTX* und *POTY*) verbunden wird. Der aus den Registern 25 und 26 ausgelesene Wert ist ein Maß für den Widerstand.

Um die gesamte Skala von 8 Bits ausnutzen zu können, muß sich der Widerstand im Bereich von 200 Ohm (nicht kleiner bitte!) bis 200 Kiloohm bewegen. Die programmtechnische Handhabung der A/D-Wandler wird im nächsten Abschnitt behandelt.

4.1.4.2 Die Verwendung von Paddles

Paddles sind nichts anderes als Potentiometer in handgreiflicher Form und eignen sich also somit hervorragend zum Anschluß an die A/D-Wandler. An den Commodore 128 können handels-

übliche Paddles angeschlossen werden. Man schließt sie einfach an die Controlports 1 oder 2 an, wo Sie sonst den Joystick anschließen können.

Da einige Bits im CIA 1 und 2 sowohl für die Tastatur als auch für die Paddle-Abfrage verantwortlich sind, ist die programmtechnische Lösung der Paddle-Abfrage nicht ganz einfach. Um nun nicht unsinnige Ergebnisse zu erlangen, schaltet man die Tastatur am besten ab, und zwar genau während des Zugriffs, da sonst die Tastatur überhaupt nicht mehr zu bedienen wäre.

Wir wollen Ihnen hier ein kleines Maschinenprogramm anbieten, daß eine komfortable Paddle-Abfrage ermöglicht. Als BASIC-Loader in Ihr Programm eingebunden wird es Ihnen gute Dienste leisten. Das Programm liegt im Bereich $CFBE bis $CFFF. Dieser Bereich wurde gewählt, da er sowohl im 128er-Modus als auch im 64er-Modus frei ist. Sie können es natürlich beliebig verschieben, Sie müssen dann lediglich die zwei Unterprogrammaufrufe an den Adressen $CFC1 und $CFD4 entsprechend ändern.

```
CFBE  SEI            ;TASTATURABFRAGE VERHINDERN
CFBF  LDA #$80       ;PARAMETER FÜR PADDLESATZ A
CFC1  JSR $CFEC      ;A/D-WERTE A1 UND A2 HOLEN
CFC4  STX $0201      ;UND SICHERSTELLEN
CFC7  STY $0202
CFCA  LDA $DC00      ;TASTEN A AUS CIA 1 HOLEN
CFCD  AND #$0C       ;BENÖTIGTE BITS FILTERN
CFCF  STA $0200      ;UND SICHERSTELLEN
CFD2  LDA #$40       ;PARAMETER FÜR PADDLESATZ B
CFD4  JSR $CFEC      ;A/D-WERTE B1 UND B2 HOLEN
CFD7  STX $0203      ;UND SICHERSTELLEN
CFDA  STY $0204
CFDD  LDA $DC01      ;TASTEN B AUS CIA 2 HOLEN
CFE0  AND #$0C       ;BENÖTIGTE BITS FILTERN
CFE2  STA $0205      ;UND SICHERRSTELLEN
CFE5  LDA #$FF       ;ALLE BITS AUSGANG IN CIA 1
CFE7  STA $DC92      ;UM TASTATURABFRAGE WIEDER
CFEA  CLI            ;ZU ERLAUBEN
CFEB  RTS            ;RÜCKKEHR INS BASIC-PROGRAMM
CFEC  STA $DC00      ;PADDLESATZ AUSWÄHLEN
CFEF  ORA #$C0       ;UND ENTSPRECHENDE BITS
```

```
CFF2  STA $DC02       ;AUF AUSGANG SETZEN
CFF4  LDX #$00        ;VERZÖGERUNGSSCHLEIFE ZUR
CFF6  DEX             ;BERUHIGUNG DES
CFF7  BNE $CFF6       ;A/D-EINGANGS
CFF9  LDX $D419       ;A/D 1 HOLEN
CFFC  LDY $D41A       ;A/D 2 HOLEN
CFFF  RTS             ;RÜCKKEHR INS HAUPTPROGRAMM
```

Hier nun der BASIC-Loader mit Beispielprogramm. Schließen Sie Ihre Paddles an und starten Sie das Programm. Schauen Sie, was geschieht.

```
10 DATA 120,169,128,32,236,207,142,1,2,140,61,3,173
20 DATA 0,220,41,12,141,0,2,169,64,32,236,207,142
30 DATA 3,2,140,4,2,173,1,220,41,12,141,5,2,169
40 DATA 255,141,2,220,88,96,141,0,220,9,192,141,2
50 DATA 220,162,0,202,208,253,174,25,212,172,26,212,96
60 FOR M=53182 TO 53247
70 READ A: POKE M,A: NEXT: REM Maschinenprogramm einlesen
80 AX=515: REM Paddle 1 am Controlport 1
90 AY=516: REM Paddle 2 am Controlport 1
100 BA=517: REM Tasten an Paddlepaar A
110 BX=513: REM Paddle 1 am Controlport 2
120 BY=514: REM Paddle 2 am Controlport 2
130 BB=512: REM Tasten von Paddlepaar B
140 SYS 53182: REM Alle Werte holen
150 PRINT PEEK(AX)" "PEEK(AY)" "PEEK(BA);
160 PRINT PEEK(BX)" "PEEK(BY)" "PEEK(BB)
170 GOTO 140
```

4.1.5 Programmierung des SID

Wir haben bereits Begriffe wie Hüllkurve und ADSR-Kontrolle angeschnitten; lassen Sie uns nun näher darauf eingehen, um Ihnen auch in Maschinensprache zu ermöglichen, den SID direkt zu programmieren.

Die Klangfarbe wird durch die Auswahl der Schwingungsform bestimmt; Filter können weiteres zur Veränderung des Klang-

bildes beitragen. Die Hüllkurve bestimmt den Tonverlauf, also die Lautstärke, die Dauer des Anschwellens etc. Folgendes Schaubild verdeutlicht die einzelnen Stadien, die ein Ton durchläuft:

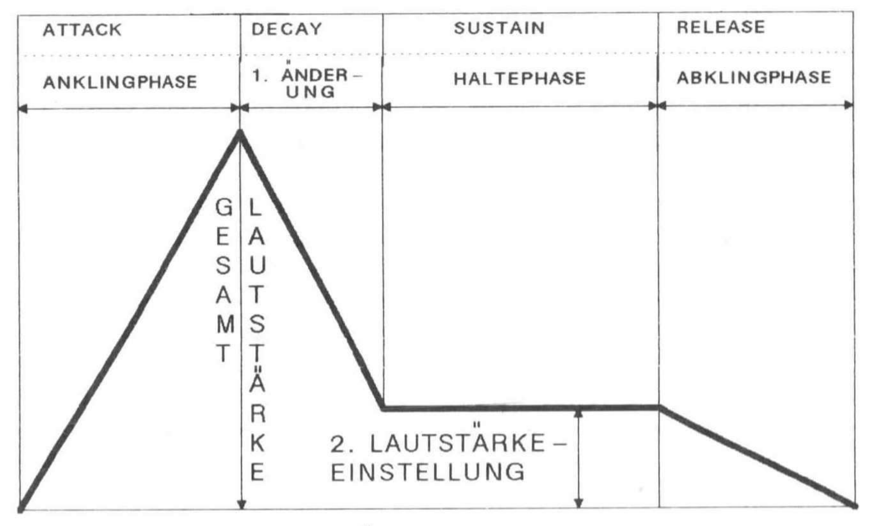


Abb. 5 (Hüllkurvenbeispiel)

Wir können auf dem Schaubild erkennen, daß ein Ton sich in vier Grundstufen unterteilen läßt: Anschwellen (*Attack*), Abfallen auf mittlere Laustärke (*Decay*), Halten (*Sustain*) und Abschwellen auf Null (*Release*). Die Dauer der einzelnen Stadien können Sie für jede Stimme einzeln bestimmen. Durch Setzen des KEY-Bits (Bit 0, Register 4 für Stimme 1) beginnt das Anschwellen des Tones, dessen Frequenz Sie zuvor definiert haben. Auch alle anderen Werte, wie Attack, Decay, Sustain und Release, müssen unbedingt vor Setzen des KEY-Bits definiert worden sein! Der Ton schwillt nun von Lautstärke Null bis auf Maximallautstärke (REG 14) innerhalb des in Attack (REG 5, Bits 4-7) definierten Zeitrahmens an. Nachdem die Maximallautstärke erreicht worden ist, fällt die Lautstärke innerhalb von Decay (REG 5, Bits 0-3) auf die zu haltende

Lautstärke Sustain (REG 6, Bits 4-7) wieder ab. Diese Lautstärke wird solange gehalten, bis das KEY-Bit wieder gesetzt wird. Ist dies geschehen, so fällt die Lautstärke innerhalb von Release (REG 6, Bits 0-3) auf Null ab. Die in Klammern angegebenen Registernummern beziehen sich auf Stimme 1, für Stimme 2 müssen Sie 7, für die Stimme 3 sogar 14 hinzuaddieren.

Die Dauer des Anschlages Attack können Sie in einem zeitlichen Rahmen von 2 ms bis 8 Sekunden definieren. Die Werte für Decay und Release liegen in einem Bereich von 6 ms bis 24 Sekunden. Diese zeitlichen Rahmen sind in 16 Stufen unterteilt worden, die Sie dieser Tabelle entnehmen können:

Wert	Attack	Decay / Release
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 Sek.
11	800 ms	2.4 Sek.
12	1 Sek.	3 Sek.
13	3 Sek.	9 Sek.
14	5 Sek.	15 Sek.
15	8 Sek.	24 Sek.

Folgendes kleines Programm soll Sie mit den Schwingungsformen und dem Tonumfang des SID 6581 vertraut machen:

```
10 S1=54272: REM Stimme 1
20 S2=54279: REM Stimme 2
30 S3=54286: REM Stimme 3
40 FL=54293: REM Filter LO-Byte
50 FH=54294: REM Filter HI-Byte
```

```
60 RS=54295: REM Resonanz+Schalter
70 PL=54296: REM Passart+Lautstärke
80 POKE S1+4,0: POKE S2+4,0: POKE S3+4,0: REM Steuerregister = 0
100 POKE S1+2,0: POKE S2+2,0: POKE S3+2,0: REM Pulsbreite auf 0
110 POKE S1+5,0: POKE S1+6,240: REM Attack/Decay Stimme 1
120 POKE RS,0: POKE PL,15: REM Keine Resonanz/Lautstärke=15
130 PRINT "Dreieck..."
140 T=16: GOSUB 300
150 PRINT "Sägezahn..."
160 T=32: GOSUB 300
170 PRINT "Rechteck..."
180 T=64: GOSUB 300
190 PRINT "Rauschen..."
200 T=128:GOSUB 300
210 PRINT "Stille..."
220 END
300 POKE S1,0: POKE S1+1,0: REM Frequenz
310 POKE S1+4,T+1: REM Ton einschalten, Schwingung definieren
320 FOR I=0 to 255: FOR J=0 TO 255 STEP 50
330 POKE S1,J: POKE S1+1,I
340 NEXT J,I
350 POKE S1+4,T: REM Ton ausschalten
360 RETURN
```

Die Zeilen 10 bis 80 sollten Sie in jedes Programm integrieren, das mit Ton arbeitet, es erleichtert die Arbeit schon sehr. Nachdem Sie das Programm abgetippt und gestartet haben, kennen Sie das Frequenzspektrum und die verschiedenen Schwingungsformen des SID. Wir wollen Ihnen aber auch gerne ein Beispiel geben, welche Auswirkungen das Verändern der Hüllkurve mitsichbringt. Hierzu übernehmen Sie der Einfachheit halber die Zeilen 10 bis 80 aus unserem Beispielprogramm und fügen folgende Zeilen an:

```
100 A=9: D=9: S=8: R=9: H=400
110 POKE S1+5,16*A+D: POKE S1+6,16*S+R
120 POKE RS,0: POKE PL,15
130 POKE S1,37: POKE S1+1,17: REM Frequenz
140 POKE S1+4,33 : REM Ton ein und Sägezahn
150 FOR I=0 TO H: NEXT
160 POKE S1+4,32: REM Ton ausschwellen lassen
```

Sie haben sicherlich schon die Bedeutung der einzelnen Variablen erkannt: *A=Attack*, *D=Decay*, *S=Sustain* und *R=Release*. Die Variable H gibt die Dauer des Haltestatus an. Verändern Sie die einzelnen Variablen, um ein Gefühl dafür zu bekommen, welche klanglichen Veränderungen Sie durch Ändern der Werte erreichen können. Achten Sie aber darauf, daß keine Variable, mit Ausnahme von H, größer als 15 werden darf! Wenn Sie von der Hüllkurve Gebrauch machen wollen, so laden Sie nach der Warteschleife, die die Dauer des Tones definiert, das Register 4 nicht mit dem Wert Null; unmittelbare Folge wäre nämlich, daß der Ton sofort verlischt. Machen Sie es so, wie in diesem Beipiel angegeben: Beim Einschalten des Tones Register 4 mit dem Wert der Schwingungsform+1 laden. Soll der Ton ausklingen, so laden Sie in das Register 4 nur noch den Wert für die Schwingungsform.

Sicherlich erlernt man die Handhabung der Programmierung am besten durch Ausprobieren. Deswegen wollen wir Sie noch mit einigen netten Beispielen beglücken, die natürlich nicht *nur so* ausgewählt wurden. Das nächste Beispielprogramm nutzt alle drei Stimmen des SID aus. Übernehmen Sie wieder die Zeilen 10-80.

```
100 A=0: D=1: S=13: R=10: H=100
110 POKE S1+5,16*A+D: POKE S1+6,16*S+R
120 POKE S2+5,16*A+D: POKE S2+6,16*S+R
130 POKE S3+5,16*A+D: POKE S3+6,16*S+R
140 POKE RS,0: POKE PL,15
150 POKE S1,37: POKE S1+1,17
160 POKE S2,154: POKE S2+1,21
170 POKE S3,177: POKE S3+1,25
180 POKE S1+4,33: POKE S2+4,33: POKE S3+4,33
190 FOR I=0 TO H: NEXT
200 POKE S1+4,32: POKE S2+4,32: POKE S3+4,32
```

Man könnte nun anhand einer Reihe von DATA-Zeilen ein ganzes Stück abspielen lassen, wozu Sie dann ein Menuett tanzen könnten, aber wir haben uns gesagt: Wer kann heutzutage noch Menuett? Also lassen wir es und beglücken Sie am Ende dieses Kapitels mit einem Stück von den Beatles.

Das nächste Beipiel dient dazu, Ihnen zu demonstrieren, wie die Frequenz eines Tones in Abhängigkeit von der Hüllkurve verändert werden kann. Hierzu wird die Stimme 3 benutzt, da man nur hier die Hüllkurve auslesen kann. Es ist auch hier recht aufschlußreich, mit den Werten in der Zeile 100 rumzuexperimentieren.

```
100 A=9: D=9: S=9: H=30
110 POKE RS,0: POKE P,15
120 POKE S3+5,16*A+D: POKE A3+6,16*S+R
130 POKE S3+4,33
140 FOR I=0 TO H: POKE S3+1,PEEK(54300): NEXT
150 POKE S3+4,32
160 FOR I=0 TO R*4: POKE S3+1,PEEK(54300): NEXT
```

Wir wollen Ihnen nun noch ein Beispiel geben für einen Geräuscheffekt, wir lassen Raumschiff Enterprise durch unser Wohnzimmer sausen:

```
100 A=15: D=0: S=8: R=13: H=8000
110 POKE RS,0: POKE PL,15
120 POKE S1,0: POKE S1+1,30
130 POKE S2,0: POKE S2+1,1
140 POKE S3,0: PKKE S3+1,100
150 POKE S1+5,16*A+D: POKE S1+6,16*S+R
160 POKE S1+4,129: POKE S3+4,23
170 FOR I=0 TO H: NEXT
180 POKE S1+4,128: POKE S3+4,16
```

Wollen Sie eine Note für den SID umsetzen, so müssen Sie die Frequenz dieser Note in die folgende Formel einsetzen:

$$F = Fre/0.06097$$

Da sich dieser Wert aus einem Low- und ein High-Wert zusammensetzt, müssen Sie den errechneten Wert noch "nachbehandeln":

$$F_l=F\ AND\ 15\ :\ F_h=INT(F/256)$$

4.2 Die Filter

Lassen Sie uns noch kurz einiges zu den Filtern sagen: Der SID verfügt über drei Filter, die Sie einzeln aber auch kombiniert verwenden können. Man kann den Oberwellengehalt einer Klangwelle (etwas anderes ist ein Ton ja nicht) durch Filter gezielt verändern. Durch den Hochpaßfilter können Sie Frequenzen unterhalb der definierten Grenzfrequenz dämpfen. Die Töne klingen dann ein wenig blechern. Das Gegenstück zum Hochpaßfilter ist der Tiefpaßfilter. Sie können es sich bestimmt schon denken. Frequenzen oberhalb der definierten Grenzfrequenz werden gedämpft. Schließlich gibt es noch den Bandpaßfilter, der lediglich ein schmales Frequenzband durchläßt, alle anderen Frequenzen werden gedämpft. Werden beispielsweise Hoch- und Tiefpaßfilter gleichzeitig eingeschaltet, so wird nur die Grenzfrequenz gedämpft, alle anderen Frequenzen bleiben unberührt. Man nennt dies auch Bandsperre.

Neben Filtertyp und Filterfrequenz kann man beim Commodore aber auch noch die sogenannte Filterresonanz einstellen. Um die Bedeutung dieses Parameters voll zu verstehen, sollte man sich den Filter als vierten Oszillator im Sound-Chip vorstellen. Bei den Filtern kann man ja ebenso wie bei den Oszillatoren eine Frequenz einstellen.

Durch den Resonanz-Wert kann man angeben, in welchem Grad der Filter selbst als Oszillator wirkt. Wenn die Resonanz auf Null eingestellt ist, so tut der Filter nichts anderes, als Frequenzen abzuschneiden (wie bereits besprochen.) Wenn man aber den Resonanz-Wert schrittweise erhöht, so fängt der Filter bei der Filterfrequenz selbst immer mehr an zu schwingen.

Der Maximalwert der Filterresonaz beträgt **15** - der Klang der durch die Filter geleiteten Oszillatoren klingt dann stark verändert und durch die Filterfrequenz beeinflußt. Man erkennt leicht, daß man mittels der Filter ein großes Spektrum an neuen Klängen hinzugewinnt, da die normalen Wellenformen auf mannigfaltige Art und Weise manipuliert werden können.

Aus der nun folgenden Registertabelle könenn Sie ersehen, welche Register des SID den Filter beeinflussen:

Nr.	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
21						freq 2	freq 1	freq 0
22	freq10	freq 9	freq 8	freq 7	freq 6	freq 5	freq 4	freq 3
23	res 3	res 2	res 1	res 0	filtex	filt 3	filt 2	filt 1
24	3 AUS	Hochp.	Bandp.	Tiefp.	Vol 3	Vol 2	Vol 1	Vol 0

4.3 Synchronisation und Ring-Modulation

Die Filter ermöglichen es uns, die Signale der einzelnen Oszillatoren gezielt zu verändern. Es gibt aber noch andere Möglichkeiten, wie wir im SID Signale eines Oszillators verfremden können: die Synchronisation und die Ring-Modulation.

Während beim Filter immer nur das Signal eines einzelnen Oszillators verfremdet werden konnte, bieten die Synchronisationen und die Ring-Modulation uns die Möglichkeit, in Abhängigkeit vom Signal zweier Oszillatoren das Signal eines der beiden Oszillatoren oder auch beider zu verändern. Man muß sich das so vorstellen, daß ein Oszillator als Tonquelle wie bisher wirkt, aber sein Signal durch das Signal eines anderen Oszillators bestimmt wird.

Bei der Ring-Modulation werden die digitalen Zahlenwerte der Schwingungen von bestimmendem Oszillator und beeinflussendem Oszillator innerhalb des SID miteinander multipliziert und durch den beeinflußten Oszillator ausgegeben. Wenn die Frequenzen beider Oszillatoren dicht beieinander liegen, entstehen sehr komplexe Wellenformen, die viele unharmonische Obertöne enthalten, so daß sie oft metallisch oder glockenähnlich klingen.

Doch nun das bereits versprochene Programm zum Abspielen des Beatles-Song "Let it be":

```
0 REM *** LET IT BE ***
4 FOR I=54272 TO 54296: POKE I,0: NEXT
10 ERSTER=54272
```

```
11 LAUTST=ERSTER+24
12 AN    =ERSTER+5
13 AUS   =ERSTER+6
14 H1    =ERSTER
15 H2    =ERSTER+1
16 DRUECK=ERSTER+4
20 POKE LAUTST,15
21 POKE AN,23
22 POKE AUS,123
30 READ HOEHE,DAUER
40 IF HOEHE=0 THEN END
50 F2=HOEHE/256: F1=HOEHE AND 255
60 POKE H2,F2: POKE H1,F1
70 POKE DRUECKM33: FOR I=0 TO DAUER*100:NEXT
80 POKE DRUEK,32: FOR I=0 TO DAUER*100: NEXT
90 GOTO 30
100 :
110 REM *** NOTEN ***
120 :
130 DATA 6430,1,6430,1,6430,3,6430,1,7217,2,5407,2
140 DATA 6430,1,6430,2,8583,3,9634,2,9634,1,10814,2
150 DATA 10814,3,9634,2,9634,2,8583,1,8583,5,10814,1
160 DATA 11457,3,10820,2,10814,2,9634,4,10814,1,9634,1
170 DATA 9634,1,8583,11,10814,1,9634,2,8534,5,10814,1
180 DATA 12860,2,14435,5,12860,1,12860,2,10814,3,6349,2
190 DATA 6349,1,6349,2,10814,9,10814,2,11457,3,10820,2
200 DATA 10814,2,9634,4,10814,1,9634,1,9634,1,8583,15
210 DATA 0,0
```

Das soll unser Kapitel über den SID abschließen. Wir hoffen, Sie haben genügend Anregungen vorgefunden, um diesen Chip nicht länger in Ihrem Rechner ruhen zu lassen. Dies gilt besonders auch für diejenigen unter Ihnen, die in Maschinensprache programmieren können und wollen, da gerade hier ungeahnte Möglichkeiten offen sind - die Maschinensprache ist schnell genug für die *unmöglichsten* Effekte. Viel Spaß!

5. Der 8563 VDC-Chip

5.1 Allgemeines über den VDC-Chip

Wie bereits in Kapitel 2 beschrieben wurde, können Sie zwei verschiedene Monitore an Ihren Commodore 128 anschließen. Der eine davon wird durch den VIC-Chip versorgt. Der andere, der RGB-Ausgang, wird durch den 8563 VDC versorgt; auf diesem Bildschirm können Sie 80 Zeichen pro Zeile darstellen. Der 80-Zeichen-Bildschirm eignet sich hervorragend für professionelle Anwendungen, die mit einem 40-Zeichen-Bildschirm nicht oder nur ungenügend möglich sind. RGB steht für *Red - Green - Blue*, was bedeutet, daß Sie die Farben Rot, Grün und Blau auf Ihrem Bildschirm darstellen können und natürlich alle damit möglichen Kombinationen. Die Farbe Weiß beispielsweise wird durch additive Farbmischung aller drei Farben erzielt - die Farbe Gelb können Sie durch Mischen von Rot und Grün erhalten. Aber keine Angst - Sie brauchen sich keinerlei Gedanken zu machen, welche Farben Sie mischen müssen, um eine bestimmte Farbe zu erhalten. Auf die Farbcodes der 16 möglichen Farben kommen wir noch zurück.

Ein wichtiger Pluspunkt des VDC-Chips ist, daß er keinen Speicher vom RAM abzwackt, um dort seinen Bildschirminhalt zu speichern. Er verfügt vielmehr über 16 KByte eigenen Speicher, in dem er sowohl Video-RAM als auch Attribut-RAM speichert. Selbst der Zeichengenerator wird in diese 16 KByte kopiert. Sollten Sie die Taste ASCII/DIN betätigen, so bemerken Sie, daß es doch einige Zeit dauert, bis der Cursor wieder voll aktionsbereit ist. Auf dem 40-Zeichen-Bildschirm vollzieht sich das Umschalten des Zeichensatzes recht schnell; auf dem 80-Zeichen-Bildschirm dauert es seine Zeit, bis alle Zeichen ihre neue Form angenommen haben. Manche Zeichen erscheinen eher in ihrer neuen Form als andere. Das liegt daran, daß die gesamten 4096 Byte aus dem ROM in das RAM des Video-Controllers kopiert werden. Stutzen Sie nun und denken: Wieso denn 4096 Byte? Es reichen doch 2048 Byte aus, um 256 Zeichen zu definieren! Sie haben natürlich recht, jedoch werden in den VDC-Speicher beide möglichen Zeichensätze kopiert, die Sie auf dem 40-Zeichen-Monitor mittels der Commodore-Taste umschalten können. Auf dem 80-Zeichen-Bildschirm können

beide Zeichensätze gleichzeitig dargestellt werden, es gibt dafür ein entsprechendes Bit im Attribut-RAM, das über den ausgewählten Zeichensatz Auskunft gibt. Da sich der Zeichensatz also im RAM des VDC befindet, hat man es natürlich leicht, das Aussehen einzelner Zeichen zu verändern, indem man einfach die entsprechenden Inhalte des RAMs verändert.

Aber wie alles Schöne hat auch diese Einrichtung des eigenen RAM seine Kehrseite: Die Adressierung des RAMs ist recht umständlich - sie muß über zwei Register des VDC-Chips indirekt erfolgen. Doch darauf wollen wir ein wenig später genauer eingehen.

Wer glaubt, es sei langweilig, diesen Chip näher zu betrachten, der täuscht sich. Dieser Chip bietet eine Flut von Manipulationsmöglichkeiten, die alle zu beschreiben, den Rahmen dieses Buches sprengen würde. Es sei jedoch jedem Tüftler geraten, diesen Chip ein wenig genauer zu betrachten, da man immer wieder neue Möglichkeiten findet. Wir wollen uns hier auf die wesentlichen, interessantesten Möglichkeiten beschränken. Die Erwartungen, die sich in Verbindung mit einem 80-Zeichen-Controller ergeben, werden sogar noch übertroffen: mit diesem Video-Controller läßt sich sogar hochauflösende Grafik mit einer Auflösung von 640 mal 200 Punkten (!!) darstellen. Hierauf gehen wir noch genaustens ein!

5.2 Die Pinbelegung

1	CCLK; Character Clock (ist auf Masse gelegt)
2	-DCLK; Dot Clock
3	HSYNC; Horizontale Synchronisation
4	CS; Systemtakt
5-6	Nicht angeschlossen
7	-CS; Chip Select
8	-RS; Register-Select (Adressleitung A0)
9	-R/W; Read-Write Auswahl
10-11	D7-D6; Datenleitungen D7-D6
12	GND;
13-18	D5-D0; Datenleitungen D5-D0
19	DISPEN; Display Enable (nicht verdrahtet)
20	VSYNC; Vertikale Synchronisation
21	DR/-W; Display-RAM READ/WRITE
22	-RES; Resetleitung (Ausgang) - Bedeutung unbekannt
23	-RES; Resetleitung (Eingang)

24	TST; Bedeutung unbekannt
25	LPEN; Light-Pen
26-33	DA0-DA7; Adresse Display-RAM
34-42	DD0-DD7; Datenleitungen Display-RAM
37	VCC; Betriebsspannung +5V
43	I; Intensity
44	B; Blue
45	G; Green
46	R; Red
47	-RAS; Row-Address-Select
48	-CAS; Column-Address-Select

5.3 Die Register des VDC-Chips

Der 8563 VDC-Chip verfügt über 37 Register, die im einzelnen folgende Bedeutung haben:

REG 0 HORIZONTAL TOTAL; (126)
In diesem Register wird die Totalanzahl der Zeichen pro Zeile minus 1 inklusive Strahlrücklauf angegeben. Dieses Register sollte mit einem 8-Bit-Wert entsprechend technischen Daten des Monitors programmiert werden.

REG 1 HORIZONTAL DISPLAYED; (80)
In diesem Register wird die Anzahl der tatsächlichen Zeichen pro Zeile programmiert. Es sind alle 8-Bit-Werte kleiner als REG 0 möglich. Der Standardwert beträgt 80.

REG 2 HORIZONTAL SYNC POSITION; (102)
In diesem Register wird der linke Rand synchronisiert. Es sind alle 8-Bit-Werte kleiner als REG 0 möglich. Wenn das Register verkleinert wird, so bewegt sich der linke Rand nach rechts; wird der Inhalt vergrößert, so bewegt sich der linke Rand entsprechend nach links.

REG 3 SYNC WIDTH; (73)
Bits 0-3 bestimmen die horizontale Sync-Puls-Breite in Zeichen. Der Wert Null läßt sich nicht programmieren.
Bits 4-7 bestimmen die vertikale Sync-Puls-Breite in Vielfachen einer Raster-Periode. Wenn Null programmiert wird, so bedeutet dies den Wert 16.

REG 4 VERTICAL TOTAL; (39)
In diesem Register wird die Anzahl der totalen Zeilen inklusive des vertikalen Strahlrücklaufes programmiert. Dieses Register sollte nach näheren technischen Daten des Monitors programmiert werden.

REG 5 VERTICAL TOTAL ADJUST; (224)
Bits 0-4 dienen der Feineinstellung von REG 4. Bits 5-7 sind immer gesetzt. Der gelesene Standardwert 224 bedeutet, daß die Bits 0-4 gelöscht sind.

REG 6 VERTICAL DISPLAYED; (25)
Hier wird die Anzahl der darzustellenden Zeilen programmiert. Jeder Wert, der kleiner als REG 4 ist, ist möglich.

REG 7 VERTICAL SYNC POSITION; (32)
Dieses Register definiert den oberen Rand des Bildschirmes. Wird der Inhalt des Registers vergrößert, so verschiebt sich der Bildschirm nach oben - entsprechend verschiebt sich der Bildschirm bei Verringerung des Registers nach unten.

REG 8 INTERLACE MODE; (252)
Bits 0-1 bestimmen den Abtastmodus. Normalerweise sind diese Bits gelöscht.
00 und 10 = Non-Interlace-Mode,
01=Interlace-Sync-Mode (Bildschirm scheint zu flackern),
11=Interlace-Sync- und Video-Mode. Probieren Sie dies einmal aus!

REG 9 CHARACTER TOTAL VERTICAL; (231)
Bits 0-4 bestimmen die Anzahl an Rasterzeilen pro Zeichen (vertikal) minus eins. Bits 5-7 sind immer gesetzt. Der Standardwert 231 steht für 7, also 7 + 1 = 8 Rasterzeilen pro Zeichen.

REG 10 CURSOR MODE/START RASTER; (160)
Bits 5-6 bestimmen den Modus des Cursors:
00=nicht blinken,
01=Cursor wird nicht angezeigt,
10=schnell blinken,
11=normal blinken

REG 11 CURSOR END SCAN LINE; (231)
Es sind nur die Bits 0-4 relevant, die anderen sind immer gesetzt. In diesem Register wird die Zeile angegeben, an der der Cursor aufhören soll. Bei Blockcursor beispielsweise beginnt der Cursor an Zeile 0 und hört an Zeile 7 auf. Beim Underline-Cursor: Start und Ende bei 7.

REG 12 DISPLAY START ADDRESS HI; (0)
In diesem Register wird das Hi-Byte des Video-RAM-Anfangs gespeichert. Normalerweise liegt das Video-RAM ab Adresse $0000 im speziellen Speicher des VDC.

REG 13 DISPLAY START ADDRESS LO; (0)
Hier wird entsprechend zum Register 11 das Lo-Byte des Video-RAMs definiert.

REG 14 CURSOR POSITION HI;
In diesem Register wird das Hi-Byte des Cursors definiert. Die Cursoradresse muß angegeben werden, da der VDC den Cursor (wenn erwünscht) selbständig blinken läßt.

REG 15 CURSOR POSITION LO;
Hier wird entsprechend zu Register 14 das Lo-Byte der Cursoradresse definiert.

REG 16 LIGHT PEN VERTIKAL;
Dieses und das folgende Register können ausschließlich gelesen werden. Die beiden höherwertigen Bits im Register 16 sind immer Null. Das Register gibt die vertikale Lightpenadresse wieder. Allerdings muß der Wert von der Software korrigiert werden, da der Kathodenstrahl sich schon um einiges weiterbewegt hat, bis die Rasterzeile erkannt wird.

REG 17 LIGHT PEN HORIZONTAL;
Entsprechend zu Register 16 befindet sich in diesem Register die horizontale Adresse des Lightpens.

REG 18 UPDATE ADRESS HI;
In diesem Register gibt man das Hi-Byte der zu manipulierenden Adresse an. Dabei ist egal, ob man Video-RAM, Attribut-RAM oder sonst eine Speicherstelle ändern will.

REG 19 UPDATE ADDRESS LO;
In Verbindung mit Register 18 wird hier das Lo-Byte der zu manipulierenden Adresse angemeldet.

REG 20 ATTRIBUT ADDRESS HI; (4)
In diesem Register wird das höherwertige Byte der Startadresse des Attributspeichers angegeben. Das Attribut-RAM definiert Farbe und Status aller auf dem Bildschirm befindliches Zeichen.

REG 21 ATTRIBUT ADDRESS LO; (0)
In Verbindung mit Register 20 wird hier das niederwertige Byte der Startadresse definiert. Im Normalmodus befindet sich das Attribut-RAM an Adresse $0400 ff.

REG 22 CHARACTER TOTAL & DISPLAYED; (120)

Bits 4-7 bestimmen die Totalanzahl der dargestellten Horizontalzeilen minus 1!(7)

Bits 0-3 bestimmen die dargestellte Anzahl an Zeilen (8). Es wird die Breite eines Zeichens definiert.

REG 23 CHARACTER DSP(V); (232)

Anzahl der dargestellten Vertikalzeilen (8); es wird die Länge eines Zeichens definiert.

REG 24 VERTICAL SMOOTH SCROLL; (32)

Bit 7: COPY-Bit; Wenn dieses Bit gesetzt ist, wird bei Beschreiben des Wordcount-Registers (REG 30) der Bereich von Block Start-adresse nach Update-Adresse kopiert. Ist dieses Bit gelöscht, wird die Update-Adresse mit Data-Register (REG 31) aufgefüllt.

Bit 6: RVS-Bit; Ist das Bit gesetzt, so wird der gesamte Bildschirm revers, d.h. umgekehrt (negiert) dargestellt. Ein gesetzter Punkt ist praktisch ungesetzt und umgekehrt.

Bit 5: CBRATE; Character Blink Rate;
Ist dieses Bit 1, so beträgt die Blinkrate bei blinkenden Zeichen 1/30 der Bildwiederholfrequenz, bei 0 1/16.

Bits 0-4: Hier kann der vertikale Rand in Rasterzeilen verschoben werden (Smooth-Scrolling)

REG 25 HORIZONTAL SMOOTH SCROLLING; (64)

Bit 7: TEXT; Ist dieses Bit gelöscht, so wird Textmodus dargestellt. Die Informationen für die Zeichen werden aus dem CHARROM geholt. Ist das Bit gesetzt, so wird Einzelpunktgrafik eingeschaltet.

Bit 6: ATR; Dieses Bit zeigt an, ob die Farbinformationen für ein Zeichen aus dem Attribut-RAM geholt werden sollen (gesetztes Bit) oder ob alle Punkte monochrom erscheinen (Farbe ist REG 26)

Bit 5: SEMI; Semi-Grafik-Betriebsart;
1: der evtl. vorhandene horizontale Zwischenraum zweier Zeichen wird in der Farbe des zuletzt dargestellten Zeichens aufgefüllt.
0: wie (1), jedoch wird der Zwischenraum mit der Hintergrundfarbe aufgefüllt.

Bit 4: DBL; Ist dieses Bit gesetzt, so erscheinen die Zeichen in doppelter Breite.
0: Pixelgröße=1-Dot-Takt
1: Pixelgröße=2-Dot-Takte

Bits 0-3: Hier kann der horizontale Rand in Rasterzeilen verschoben werden (Smooth-Scrolling).

REG 26 FORGRND/BACGRND; (240)
Bits 0-3 bestimmen die Hintergrundfarbe.
Bits 4-7 bestimmen die Vordergrundfarbe für Grafik- oder Monochrommodus.

REG 27 ADDRESS INCREMENT ROW; (0)
Es wird definiert, wieviel Bytes zum Video-RAM bei jeder Spalte zu addieren sind. Normalerweise ist dies Null. Definiert man die Zeichenbreite beispielsweise um (und somit die Anzahl Zeichen/Zeile), so muß dieser Wert umprogrammiert werden.

REG 28 CHARACTER BASIS ADDRESS; (47)
Bits 5-7 bestimmen die Basis des Zeichengenerators bzw. die Adreßbits 13 bis 15; der Zeichengenerator kann also lediglich in 8K-Schritten verschoben werden.
Bit 4: RAM; Dieses Bit definiert den RAM-Typ:
1: 4164; 0: 4416

REG 29 UNDERLINE SCAN LINE; (231)
Bits 0-4 geben die Zeile an, in der unterstrichen werden soll. Standardwert ist 8. Beispielsweise könnte man durch Ändern dieses Registers aus dem Unterstreichen ein Über- oder Durchstreichen machen.

REG 30 WORD COUNT;
In dieses Register schreibt man die Anzahl der Zeichen, die an die Update-Adresse geschrieben werden sollen bzw. wenn das COPY-Bit gesetzt ist, wieviele Bytes kopiert werden sollen.

REG 31 DATA;
In dieses Register schreibt man die Daten, die an eine Speicherstelle gespeichert werden sollen. Soll eine Speicherstelle ausgelesen werden, so muß man sich den Wert aus diesem Register holen.

REG 32 BLOCK START ADDRESS HI;
In diesem Register (und dem folgenden) definiert man die Startadresse des Blocks, der kopiert werden soll.

REG 33 BLOCK START ADDRESS LO;
Entsprechend zu Register 32 wird hier das niederwertige Byte der Startadresse definiert.

REG 34 DISPLAY ENABLE BEGIN; (125)
Anzahl der Zeichen von Beginn der dargestellten Zeile bis zur positiven Flanke des Display-Enable-Pins.

REG 35 DISPLAY ENABLE END; (64)
Wie REG 34, jedoch bis zur negativen Flanke.

REG 36 DRAM REFRESH RATE; (245)
Bits 0-3 geben die Rate an, in der der Speicher des VDC aufgefrischt werden muß (Refresh-Zyklen pro Bildschirmzeile).

Die Werte in Klammern geben die Normalwerte an, die sich normalerweise (nach Warmstart) in den Registern befinden.

5.4 Allgemeines zu den VDC-Registern

Auf alle Register näher einzugehen, wäre ebenso nutzlos wie unsinnig. Am besten erkennt man, was die einzelnen Register alles bewirken, indem man sie einfach mit Werten beschreibt und sieht, was geschieht. Man darf diese Register nicht alle als nutzbringende Register für den Programmierer sehen, so wie es beispielsweise beim VIC oder SID der Fall ist. Hier sind vielmehr eine Menge der Register ausschließlich zur Bilddarstellung und -synchronisation vorhanden, die man praktisch kaum oder gar nicht ändern darf oder sollte.

Die Basisadresse des 80-Zeichen-Video-Controllers ist $D600. Übrigens als kleiner Tip: Zumindest bei unserem Prototyp konnte man den VDC auch vom 64er-Modus aus manipulieren: Also 80-Zeichen-Modus auch im 64er-Modus möglich! Neben der Möglichkeit, im 2-Mhz-Modus die Programme zu fahren, eine weitere kleine Lücke in der Kompatibilität des 64er-Modus.

Jetzt kann man beim VDC allerdings nicht die verschiedenen Register so einfach adressieren wie beim VIC oder SID, indem man die Registernummer einfach zur Basisadresse addiert. Beim VDC erfolgt die Registermanipulation relativ, d.h. man muß dem Controller mitteilen, welches Register man auslesen oder beschreiben will, und kann dann in einem fixen Register diese Manipulation (Lesen oder Schreiben) vornehmen. Sicherlich eine recht komplizierte Methode, aber man gewöhnt sich schnell dran. Will man beispielsweise nur ein Byte im Video-RAM ändern, so muß man diese Speicheradresse relativ über die Register ansprechen (da sie ja keine direkt ansprechbare Adresse ist) und diese Register wiederum relativ auswählen.

Doch kommen wir zur Sache: Ansprechen kann man den VDC ausschließlich an den Adressen $D600 und $D601. Will man

beispielsweise ein Register auslesen, so muß man die Registernummer in Adresse $D600 schreiben. Der VDC holt dann den aktuellen Inhalt des Registers in die Adresse $D601, die man dann auslesen kann. Will man ein Register beschreiben, so verfährt man adäquat: Man teilt über Adresse $D600 die Registernummer mit und schreibt dann an Adresse $D601 den neuen Registerwert.

Adresse $D600

(Schreiben)	---	---	R5	R4	R3	R2	R1	R0
(Lesen)	Status	LP	VBLANK	--	--	--	--	--

Adresse $D601

(Schr/Lesen)	D7	D6	D5	D4	D3	D2	D1	D0

Schreibt man an die Adresse $D600, so geschieht dies ausschließlich zur Registerauswahl. Hierzu dienen die Bits 0 bis 5. Man kann die Adresse $D600 allerdings auch auslesen, man erhält dann einen Statusreport des VDC. Bit 7, das Statusbit, gibt an, ob der VDC bereits mit seiner letzten Aktion fertig ist. Ist das Bit gesetzt, so ist der Video-Controller noch nicht fertig, und man sollte warten, bis er einem grünes Licht gibt, da sonst mit Sicherheit Daten verschluckt werden. Allerdings ist diese Abfrage nur in Maschinensprache von Nöten, da BASIC viel zu langsam ist, um das Statusbit auch nur einmal nicht gesetzt zu erwischen. Wollen wir beispielsweise das DATA-Register des VDC in Maschinensprache beschreiben, so sollte dies so aussehen:

```
     LDA #$1F  ;DATA-Register
     STA $D600 ;auswählen
Wait BIT $D600 ;Teste Statusbit
     BPL Wait  ;Noch gesetzt, dann nicht fertig
     LDA #$21  ;ASCII-Code für "!"
     STA $D601 ;und hineinschreiben
     RTS       ;Rücksprung
```

Sie sehen, wie in dieser kleinen Routine nach Beschreiben der Adresse $D600 gewartet wird, bis das Statusbit zurückgenommen wird. Erst dann wird der Wert für das DATA-Register übermittelt. Gegebenenfalls sollte man nach Beschreiben der Adresse $D601 noch einmal eine Warteroutine einbauen; jedoch

hängt dies stark vom Programm ab. Werden beispielsweise nach Beschreiben von $D601 erst irgendwelche Berechnungen ausgeführt, bevor der VDC wieder angesprochen wird, so braucht man nicht zu warten.

Das Bit 6 an Adresse $D600 ist für den Lightpen reserviert und soll uns momentan nicht weiter interessieren. Das Bit 5 teilt uns mit, ob sich der Kathodenstrahl gerade im Rücklauf befindet (Bit ist gesetzt) oder nicht. Dies kann man sich beispielsweise für irgendwelche Synchronisationszwecke zu Nutze machen. Die restlichen Bits sind nicht belegt.

Beim Beschreiben der Adresse $D600 wird die Registerauswahl vorgenommen. An Adresse $D601 werden immer nur Daten transferiert.

Um beispielsweise den Wert des DATA-Registers zu erhalten, ist folgendes kleines Maschinenprogramm nötig:

```
     LDA #1F   ;DATA-Register
     STA $D600 ;Register anmelden
Wait BIT $D600 ;Statusbit noch gesetzt?
     BPL Wait  ;noch nicht fertig
     LDA $D601 ;Hole aktuellen Inhalt
```

Von BASIC können wir natürlich auch Manipulationen am VDC vornehmen. Nur soviel vorweg: Wegen der Geschwindigkeit gibt es dann und wann schon einmal Probleme, darum seien Sie nicht verärgert, wenn nicht alles sofort klappt. Das Beschreiben und Laden des DATA-Registers sähe in BASIC wie folgt aus:

```
10 A=DEC("D600"): D=A+1: REM Basisadresse VDC
20 POKE A,31: PRINT PEEK(D): REM Registerinhalt holen
30 POKE A,31: POKE D,33: REM Register beschreiben
```

Sicherlich wollen Sie aber jetzt wissen, wie man eine Adresse auf dem Bildschirm manipulieren kann. Wir wissen, daß das Video-RAM an Adresse $0000 beginnt und exakt 2000 Zeichen umfaßt. Um eine Adresse im RAM zu manipulieren, muß man diese erst im Update-Register definieren, egal ob man lesen oder schreiben will. Am besten wird die Vorgehensweise deut-

lich, indem wir uns einmal ein kleines BASIC-Programm anschauen, das natürlich im 128er-Modus läuft!

```
10 A=DEC("D600"): D=A+1
20 POKE A,18: POKE D,0: REM Update-Adresse Hi-Byte
30 POKE A,19: POKE D,0: REM Update-Adresse Lo-Byte
40 POKE A,31: POKE D,1: REM Eine 1 für "A"
50 POKE A,30: POKE D,1: REM Zeichenzähler setzen
```

Aus diesem Beispiel wird gleich mehreres deutlich: Erst einmal die Reihenfolge, die Sie in Ihren Programmen auch unbedingt einhalten sollten. Zuerst die Update-Adresse angeben, dann das Zeichen, das dargestellt werden soll und als letztes den Zeichenzähler beschreiben. Sobald das Zeichenzählregister beschrieben wird, beginnt der Video-Controller nämlich mit der Aktion. Das Ergebnis ist natürlich entsprechend unbefriedigend, wenn man dann die Update-Adresse noch nicht definiert hat.

Unbefriedigend wird aber auch diese kleine Routine für Sie sein, unabhängig davon, ob Sie sich im FAST- oder im SLOW-Modus befinden. Sie können das noch mehr verdeutlichen, indem Sie folgende Zeilen in das Programm einbauen:

```
5 PRINT CHR$(19);"  "
60 GETKEY A$: RUN
```

Immer wenn Sie eine Taste drücken, werden die ersten beiden Stellen am Bildschirm gelöscht. Danach wird der Videocontroller "aufgefordert", ein "A" an die erste Bildschirmposition zu setzen. Auf diese Weise können wir kontrollieren, ob auch wirklich ein A an die richtige Stelle gesetzt wird.

Wenn wir das Programm starten, so stellen wir fest, daß das Ergebnis nicht unseren Erwartungen entspricht. Das A hüpft von links nach rechts, es wird also nicht immer an die korrekte Stelle gesetzt. Manchmal wird gar ein "@" anstatt des A's auf dem Bildschirm angezeigt.

Leider läßt sich hier kein besseres Ergebnis erzielen - in BASIC ist dies, so scheint es zumindest, schier unmöglich. Wir haben verschiedene Verfahren ausprobiert, und keins führte zum Erfolg. BASIC ist hierzu einfach zu langsam.

Doch was man nicht in BASIC lösen kann, das sollte doch zumindest in Maschinensprache möglich sein. Lassen Sie uns deshalb ein kleines Maschinenprogramm besprechen, das genau dasselbe tun soll, wie unser BASIC-Programm.

Hierzu erst einmal das Assembler-Listing dieser Routine, die ebenfalls ein "A" auf dem Bildschirm bringen soll:

```
00D00    8E 00 D6  STX $D600
00D03    2C 00 D6  BIT $D600
00D06    10 FB     BPL $0D03
00D08    8D 01 D6  STA $D601
00D0B    60        RTS
00D0C    A2 12     LDX #$12
00D0E    A9 00     LDA #$00
00D10    20 00 24  JSR $0D00
00D13    E8        INX
00D14    20 00 24  JSR $0D00
00D17    A2 1F     LDX #$1F
00D19    A9 01     LDA #$01
00D1B    20 00 24  JSR $0D00
00D1E    CA        DEX
00D1F    4C 00 24  JMP $0D00
```

Diese kleine Maschinenroutine, die Sie wohl am einfachsten mit dem eingebauten Monitor eingeben, testen Sie am besten, indem Sie folgendes BASIC-Programm verwenden:

```
10 PRINT CHR$(147);
20 SYS DEC("0D0C"): GETKEY A$: RUN
```

Starten Sie einmal dieses Programm: Das Ergebnis wird wohl auch Sie überraschen. Die Positionierung wird nun richtig vorgenommen, allerdings werden zwei anstatt eines A's dargestellt. Der VDC stellt also Wordcount + 1 mal das definierte Zeichen dar; dies allerdings sehr sorgsam und an der korrekten Adresse. Wollten wir mehr als ein A darstellen, so wären wir nun aus

dem Schneider. Da unser Problem aber damit begann, ein A darzustellen, müssen wir noch weiter nach einer Lösung forschen. Übrigens werden 256 Zeichen dargestellt, wenn man das Wordcount-Register mit dem Wert Null lädt.

Die Lösung ist (wieder einmal?) recht einfach: Will man ein beliebiges Zeichen darstellen, so muß man nach der Definition von Update-Adresse und DATA-Register nicht das Wordcount-Register beschreiben, sondern lediglich die Update-Adresse mit einem neuen Wert laden bzw. dieses Register auslesen - dann klappt's.

Um dies einmal auszuprobieren, brauchen wir unsere Maschinenroutine lediglich ab Adresse $00D1E zu verändern:

```
00D1E    A2 12     LDX #$12
00D20    4C 00 24  JMP $0D00
```

Sie sehen, es ist egal, mit welchem Wert wir das Update-Register beschreiben. Die Routine liegt im Ausgabe-Buffer für die RS232. Jetzt ändern wir die Maschinenroutine noch leicht ab, dann können wir an jede beliebige Bildschirmposition jedes beliebige Zeichen schreiben, auch in BASIC.

```
10 REM ================================================
20 REM     BASIC-LOADER FÜR 80-ZEICHEN-POKE-ROUTINE
30 REM ================================================
40 :
50 FOR I=0 TO 36
60 : READ X
70 : POKE DEC("D00")+I,X
80 : S=S+X
90 NEXT
100 IF S<>2850 THEN PRINT "*** Fehler in DATAs ***": END
110 :
120 DATA 142,0,214,44,0,214,16,251,141,1,214,96,162,18,169,0
130 DATA 32,0,13,232,169,0,32,0,13,162,31,169,1,32,0,13
140 DATA 162,18,76,0,13
150 :
```

```
160 REM *** Ausprobieren ***
170 :
180 PRINT CHR$(147);
190 SYS DEC("DOC"): GETKEY A$: GOTO 180
```

Und endlich funktioniert das lang ersehnte Programm, auch wenn es nicht *rein* in BASIC realisierbar ist. Vielleicht jedoch findet ja irgendwann jemand mal einen Algorithmus, der auch in BASIC funktioniert und es zuläßt, auf dem 80-Zeichen-Bildschirm Manipulationen durchzuführen.

Wie bereits erwähnt, können Sie mit dieser Routine an jeder beliebigen Stelle des Bildschirmes jedes beliebige Zeichen darstellen lassen. Dazu müssen Sie an Adresse $0D0F das Hi-Byte der Adresse POKEn, an Adresse $0D15 das Lo-Byte und an Adresse $0D1C das Zeichen, das hineingePOKEt werden soll. Probieren Sie hierzu einfach folgendes Beispielprogramm einmal aus:

```
10 REM ==================================================
20 REM      BEISPIELPROGRAMM FUER POKE-ROUTINE
30 REM ==================================================
40 :
50 LO=DEC("D15"): HI=DEC("D0F"): PO=DEC("D1C")
60 FOR I=0 TO 1999
70 : POKE LO, I AND 255 : REM LO-BYTE POKEN
80 : POKE HI, I/256     : REM HI-BYTE POKEN
90 : POKE PO, I AND 255 : REM BEISPIELSWEISE
100 : SYS DEC("DOC")
110 NEXT
120 GETKEY A$
```

Aber wir wollen ja nicht immer nur ein Zeichen darstellen, manchmal wäre es recht praktisch, wenn wir beispielsweise (unter Zuhilfenahme des Wordcount-Registers) gleich 80 Zeichen darstellen könnten, um eine Zeile zu löschen oder ähnliches. Aber auch hier kann es natürlich vorkommen, daß der VDC zuschlägt und ein Zeichen zuviel darstellt; dies kann schon peinlich werden, nehmen Sie als Beispiel mal eine Textverarbeitung, die diesen Fehler hätte: Sie wären sicherlich sehr verärgert.

Natürlich mußte dieser Fehler auch im Betriebssystem ausgebügelt werden. Man hat sich hier etwas zwar recht Einfaches aber umso Wirkungsvolleres einfallen lassen. Man hat die Startadresse des zu füllenden Bereichs, man weiß ebenfalls, wieviele Zeichen dargestellt werden sollen: also kann man ausrechnen, welches die letzte zu beschreibende Stelle ist. Lassen Sie den Video-Controller einfach Anzahl-1 Zeichen auffüllen. Nach der Aktion können wir anhand der Update-Adresse (die der Video-Controller selbständig inkrementiert) feststellen, ob er ein Zeichen zuviel dargestellt hat - dann sind wir fertig - ansonsten müssen wir noch ein Zeichen darstellen lassen. Diese Methode ist immer noch schneller, als wenn wir jedes Zeichen extra setzen. Lassen Sie uns auch von einer Betriebssystemroutine Gebrauch machen, die bei gesetzter Update-Adresse und gesetztem DATA-Register dieses Zeichen so oft ausgibt, wie der Wert im <Akku> anzeigt. Diese Routine befindet sich an der Adresse $C53E - die errechnete Endadresse muß man in $0A3C/0A3D ablegen. Wir wollen die Routine an die bereits bestehende anhängen:

```
00D25    A2 12       LDX #$12
00D27    A9 00       LDA #$00
00D29    20 00 0D    JSR $0D00
00D2C    8D 3D 0A    STA $0A3D
00D2F    E8          INX
00D30    A9 00       LDA #$00
00D32    20 00 0D    JSR $0D00
00D35    8D 3C 0A    STA $0A3C
00D38    A9 00       LDA #$00
00D3A    A2 1F       LDX #$1F
00D3C    20 00 0D    JSR $0D00
00D3F    A9 00       LDA #$00
00D41    18          CLC
00D42    48          PHA
00D43    6D 3C 0A    ADC $0A3C
00D46    8D 3C 0A    STA $0A3C
00D49    90 03       BCC $0D4E
00D4B    EE 3D 0A    INC $0A3D
00D4E    68          PLA
00D4F    4C 3E C5    JMP $C53E
```

Sie können den BASIC-Loader (s.o.) durch folgende DATA-Zeilen ergänzen:

```
150 DATA 162,18,169,0,32,0,13,141,61,10,232,169,0,32,0,13
160 DATA 141,60,10,169,0,162,31,32,0,13,169,0,24,72,109,60
170 DATA 10,144,3,238,61,10,104,76,62,197
```

Ferner sollten die Zeilen 50 und 100 lauten:

```
 50 FOR I=0 TO 81
100 IF S<>5859 THEN PRINT "*** Fehler in DATAs ***": END
```

Das Hi-Byte der Startadresse speichern Sie an Adresse $0D28, das Lo-Byte an Adresse $0D31. Das Füllzeichen müssen Sie an Adresse $0D39 POKEn, die Anzahl an Adresse $0D40! Beispiel:

```
POKE DEC("0D28"),0: POKE DEC("0D31"),0: REM ADRESSE
POKE DEC("0D39"),33: REM FÜLLZEICHEN
POKE DEC("0D40"),79: REM FÜLLMENGE-1
SYS DEC("0D25"):     REM AUFRUF DER ROUTINE
```

Nachdem Sie dies eingegeben haben, wird die erste Zeile mit Ausrufungszeichen aufgefüllt.

Wie bereits erwähnt, können Sie genauso, wie wir jetzt den Bildschirminhalt geändert haben, auch das Attribut-RAM ändern. Wollen Sie beispielsweise die erste Zeile blinkend weiß darstellen, so müssen wir das Attribut-RAM mit $1F = 31 füllen. Dies wollen wir einmal tun, indem wir folgende Zeilen eingeben:

```
POKE DEC("0D28"),8: POKE DEC("0D31"),0: REM ATTRIBUT-RAM
POKE DEC("0D39"),31: REM FÜLLZEICHEN
POKE DEC("0D40"),80: REM FÜLLMENGE
SYS DEC("0D25"):     REM AUFRUF DER ROUTINE
```

5.4.1 Der Zeichensatz

Sie können es sich bestimmt schon denken: Den Zeichensatz können Sie auf ähnlich einfache Art und Weise variieren. Sie müssen nur bedenken, daß pro Zeichen 16 Byte im RAM defi-

niert werden. Acht Byte werden aus dem CHARROM kopiert, acht weitere Nullbyte werden aus VDC-internen Gründen angehangen. Der Zeichensatz beginnt beim VDC an Adresse $2000. Wollen Sie ein Zeichen auslesen oder verändern, so finden Sie es also an Adresse:

*2*4096 + <Code>*16*

Anders als beim VIC können beim VDC beide duch <Shift>/<Commodore> umschaltbare Zeichensätze gleichzeitig dargestellt werden, da diese sich auch beide gleichzeitig im Speicher des VDC-RAMs befinden. Zusätzlich sind noch die reversen Zeichen definiert, obwohl dies eigentlich gar nicht nötig gewesen wäre, da es ein Bit im Attribut-RAM gibt, das ein Zeichen revers darstellen läßt. Diese beiden Möglichkeiten kann man ausschöpfen, wenn man sich zusätzliche Zeichen definieren will. Beispielsweise könnte man in einer Textverarbeitung alle möglichen mathematischen Sonderzeichen definieren, ohne merkliche Einbußen hinnehmen zu müssen. Voraussetzung ist allerdings, daß man beispielsweise geschickt mit dem Revers-Bit programmiert, um diese Zeichen auch wirklich einsparen zu können; dies sind immerhin 128 Zeichen!

Die Speicherbelegung des VDC-RAMs sieht also wie folgt aus:

$0000-$07CF:Video-RAM
$0800-$0FCF:Attribut-RAM
$2000-$3FFF:CHARROM (Zeichensatz)

5.4.2 Das Attribut

Das Attribut eines jeden Zeichens setzt sich aus mehreren Kriterien zusammen: Erst einmal dem RGB-Signal, also ob Rot, Grün oder Blau aktiv sind (bei Weiß beispielsweise sind hier alle Bits gesetzt), dann dem Helligkeitssignal, das festlegt, ob die Farbe heller oder dunkler erscheint. Dann gibt es noch ein Bit, das anzeigt, ob das entsprechende Zeichen blinken soll, ein Bit zum Unterstreichen, ein Bit zur Reversdarstellung und ein Bit für den Alternate-Zeichensatz. Sie sehen schon, daß man beim VDC eigentlich die reversen Zeichen gar nicht hätte definieren müssen, da ein entsprechendes Bit im Attribut-RAM vorhanden

ist. Damit aber nicht unnötige Kalkulationen notwendig werden, hat man den reversen Zeichensatz einfach übernommen.

Um aber wieder auf das eigentliche Attribut-RAM zurückzukommen: Die acht Bits eines Attributbytes setzen sich wie folgt zusammen:

ALT RVS UL FLASH R G B I

ALT steht für ALTernate. Anders als beim VIC können Sie beim VDC alle beide Zeichensätze, die Sie durch die Tastenkombination <Shift>/<Commodore> umschalten können, auf einmal darstellen. Ist der zweite Zeichensatz ausgewählt, so wird das ALT-Bit im Attribut-RAM gesetzt.

RVS steht für ReVerS und bedeutet: Dieses Zeichen soll revers (umgekehrt) dargestellt werden. Wie bereits erwähnt, wird von diesem Bit direkt (leider) kein Gebrauch gemacht. Bei professioneller Software wird man den Raum der reversen Zeichen sicherlich sinnvoller nutzen.

UL steht für UnderLine. Ist dieses Bit gesetzt, so wird das entsprechende Zeichen unterstrichen, und zwar in der Rasterzeile, die in REG 29 definiert ist; normalerweise ist dies Zeile 7.

FLASH steht für Blinken. Wenn man dieses Bit setzt, so blinkt das durch dieses Attribut-Byte definierte Zeichen. Dabei wird die Farbe beibehalten und ggf. unterstrichen.

R steht für *R*ot, *G* für *G*rün und *B* für *B*lau. Das Farbsignal setzt sich aus den gesetzten und gelöschten Bits zusammen. Zusätzlich gibt es noch ein Luminenzsignal *I*, mit dem man die Helligkeit bestimmen kann; gesetztes Bit bedeutet hell.

Hier nun eine Tabelle der 15 möglichen Farb- und Luminenzkombinationen:

R	*G*	*B*	*I*	*Farbe*
0	*0*	*0*	*0*	*Schwarz*
0	*0*	*0*	*1*	*Dunkelgrau*
0	*0*	*1*	*0*	*Blau*
0	*0*	*1*	*1*	*Hellblau*
0	*1*	*0*	*0*	*Grün*
0	*1*	*0*	*1*	*Hellgrün*
0	*1*	*1*	*0*	*Beige-Blau*
0	*1*	*1*	*1*	*Zynober*
1	*0*	*0*	*0*	*Rot*
1	*0*	*0*	*1*	*Hellrot*
1	*0*	*1*	*0*	*Lila*
1	*0*	*1*	*1*	*Violett*
1	*1*	*0*	*0*	*Braun*
1	*1*	*0*	*1*	*Gelb*
1	*1*	*1*	*0*	*Hellgrau*
1	*1*	*1*	*1*	*Weiß*

Zum Attribut-RAM ist sonst nichts weiteres zu sagen, da die Bedeutung und die Programmierung durch die acht Bits festgelegt sind.

5.5 Die Nutzung der VDC-Register

Wie bereits erwähnt, stehen durch die 37 VDC-Register eine Flut an Manipulationsmöglichkeiten am 80-Zeichen-Video-Controller zur Verfügung. Wir wollen Ihnen anhand einiger Beispielprogramme diese Nutzung nahelegen und demonstrieren. Für besonders sinnvoll halten wir die Möglichkeit, 28 Zeilen auf dem Bildschirm darzustellen (anstatt 25), sowie die Möglichkeit, die hochauflösende Grafik mit einer Auflösung von 640 mal 200 Punkten zu verwenden. Wir wollen uns dann auch auf diese Dinge konzentrieren.

Zuerst einmal ein Programm, das sehr dienlich ist, wenn man die Welt der VDC-Register ein wenig erforschen und durcheinander bringen will. Beim Ausprobieren wird es nicht selten vorkommen, daß Sie auf Ihrem Bildschirm nur noch Unsinn sehen. Dann haben Sie den Controller so weit verwirrt, daß er

nicht mehr im Stande ist, ein vernünftiges Bild darzustellen. Am besten ist es, Sie betätigen dann die <RUN/STOP>/<RESTORE>-Taste. Je nachdem kann es auch vorkommen, daß der Zeichengenerator überschrieben wurde. Dazu betätigen Sie am besten die <ASCII>/<DIN>-Taste, der Zeichensatz wird dann wieder zurückkopiert.

Das Programm zeigt Ihnen immer die aktuellen Registerinhalte auf dem Bildschirm an und fragt Sie dann nach Registernummer und -inhalt. Nachdem Sie die Werte eingegeben haben, können Sie die Reaktion direkt auf dem Bildschirm verfolgen (sofern eine Reaktion erfolgt). Es werden Ihnen wieder die aktuellen Registerinhalte angezeigt.

```
10 REM *** Austesten der VDC-Register ****
20 :
30 A=DEC("D600"): D=A+1
40 PRINT CHR$(147)"Aktuelle Registerinhalte -"
50 FOR I=0 TO 37
60   POKE A,I: C=PEEK(D)
70   PRINT "#";I;RIGHT$(HEX$(C),2),
80 NEXT I
90 PRINT: PRINT
100 INPUT "Register, Wert --- ";RE,WE
110 POKE A,RE: POKE D,WE: GOTO 40
```

5.5.1 Smooth-Scrolling

Smooth-Scrolling ist gleichbedeutend mit Soft-Scrolling. Selbst darauf brauchen Sie beim VDC nicht zu verzichten. Genauso wie beim VIC kann man auch beim VDC den Bildschirm vertikal oder horizontal in Rasterzeilen-Einheiten verschieben. Hierzu dienen beim VDC die Register 24 (Bits 0-4) und 25 (Bits 0-3). Allerdings anders als beim VIC-Chip braucht man hier keinerlei Einbußen bezüglich der Zeichen- oder Zeilenzahl hinzunehmen. Der VDC ist zwar nicht für Spiele geeignet - zwar hat er eine sehr gute Auflösung, aber er ist durch seine recht komplizierte Adressierungstechnik viel zu langsam - jedoch kann man mit dem Smooth-Scrolling einige andere sinnvolle Effekte erzielen. Hier ist nun ein kleines Demoprogramm,

das Ihnen die Wirkung des Smooth-Scrollings auch auf dem 80-Zeichen-Schirm demonstriert:

```
10 REM *** Demoprogramm für Smooth-Scrolling ***
20 A=DEC("D600"): D=DEC("D601")
30 VE=24: HO=25
40 PRINT CHR$(147)CHR$(27);"M"; : REM BILDSCHIRM CLR UND SCROLLEN
   AUS
50 A$="Hallo C128-Fans!"
60 FOR I=0 TO 24
70   PRINT A$
80 NEXT
90 :
100 FOR I0=0 TO 6
110   POKE A,VE: V=PEEK(D) AND 240 OR I0
120   POKE A,VE: POKE D,V
130   FOR I1=1 TO 20: NEXT
140   POKE A,HO: H=PEEK(D) AND 240 OR I0
150   POKE A,HO: POKE D,H
160   FOR I1=1 TO 20: NEXT
170 NEXT
180 GOTO 100
```

Sollte Ihnen dies zu schnell oder nicht schnell genug gehen, so ändern Sie entsprechend die Warteschleifen in den Zeilen 130 und 160.

Ist das Bit 3 gelöscht, so werden 25 Zeilen dargestellt und das folgende (bzw. vorangehende) RAM in den Bildschirm gescrollt. Setzen Sie das Bit 3, so werden nur noch 22 Zeilen dargestellt und Sie können die drei letzten Zeilen des Bildschirmes mittels Smooth-Scrolling in den Bildschirm scrollen.

5.5.2 Blockweises Kopieren

Wenn der Controller so schwierig zu adressieren ist, wie klappt dann das Scrolling auf dem Bildschirm so schnell? Vielleicht hat sich das der eine oder andere Leser bereits (zu recht) gefragt. Die Lösung ist recht einfach: Der VDC ist so intelligent, selbständig ganze Blöcke in seinem Speicher zu verschieben. Wenn

man dies über die relative Adressierung machen müßte, so würde dies auch in der Tat einiges an Zeit in Anspruch nehmen.

Will man, daß der VDC einen Speicherbereich verschiebt, so muß man ihm dies über das COPY-Bit (Bit 7 in REG 24) mitteilen. Ist das Bit gesetzt, so wird kopiert, ansonsten wird aufgefüllt. Die Startadresse des zu kopierenden Blocks wird in den Registern 32 und 33 definiert; die Zieladresse des Kopiervorganges muß man im Update-Register (REG 18 und 19) definieren; der Kopiervorgang beginnt, wenn man das Wordcount-Register beschreibt. Hier wird auch die Anzahl der zu kopierenden Zeichen angegeben.

Achtung! Es wird die tatsächlich zu kopierende Anzahl angegeben! Wollen Sie beispielsweise die erste Textzeile auf dem Bildschirm um eine Zeile nach unten kopieren und das Attribut beibehalten, so müssen Sie in zwei Schritten erst die Textzeile und dann das Attribut kopieren. Wir wollen mit unserem Beispielprogramm ein SCROLLEN-ABWÄRTS realisieren - in BASIC natürlich recht langsam, in Maschinensprache allerdings entsprechend schnell.

```
 10 REM *** Demo-Programm für Kopieren ***
 20 A=DEC("D600"): D=DEC("D601")
 30 POKE A,24: C=PEEK(D): REM *** Inhalt Register 24
 40 POKE A,24: POKE D,C OR 128: REM *** COPY-BIT setzen
 50 FOR Z=24 TO 0 STEP -1
 60   AQ=Z*80: AZ=AQ+80: REM *** Quelle und Ziel
 70   POKE A,18: POKE D,AZ/256: POKE A,19: POKE D,AZ AND 255
 80   POKE A,32: POKE D,AQ/256: POKE A,33: POKE D,AQ AND 255
 90   POKE A,30: POKE D,79: REM *** Text kopieren
100   AQ=2048+AQ: AZ=2048+AZ: REM *** Attribut-Adresse
110   POKE A,18: POKE D,AZ/256: POKE A,19: POKE D,AZ AND 255
120   POKE A,32: POKE D,AQ/256: POKE A,33: POKE D,AQ AND 255
130   POKE A,30: POKE D,79: REM *** Attribut kopiert
140 NEXT
150 PRINT CHR$(19);CHR$(27);"D"; : REM *** Erste Zeile löschen
160 POKE A,24: POKE D,C: REM *** COPY-BIT wieder löschen
```

Diese Routine tut zwar nichts anderes als die ESC-Sequenz CHR$(27);"W", jedoch wird hierdurch die Funktionsweise des Blockkopierens deutlich.

5.5.3 Vorder- und Hintergrundfarbe

Die Hintergrundfarbe des 80-Zeichen-Bildschirmes können Sie im Register 26 (Bits 0-3) definieren. Die Vordergrundfarbe hat im Grafikmodus Wirkung und - vorausgesetzt das ATR-Bit im Register 25 ist nicht gesetzt - auch im Textmodus.

Die Definition des Registers:

```
POKE DEC("D600"),26
POKE DEC("D601"),<Vordergrund>*16 + <Hintergrund>
```

5.5.4 Der Cursormodus

Sie können das Aussehen des Cursors weitgehend selbst bestimmen: Sie können ihn ganz auschalten, schnell oder langsam blinken lassen und ihn als Block- oder als Underline-Cursor definieren. Sie können diese Definitionen zwar auch über die ESC-Sequenzen vornehmen, jedoch gibt es Situationen, wo dies nicht möglich ist - beispielsweise in Maschinensprache. Der Cursormodus wird im Register 10 bestimmt. Ferner wird im Register 10 noch angegeben, an welcher Rasterzeile der Blockcursor beginnen soll. Durch Anfangs- und Endzeile des Blockcursors können Sie aus dem Block beispielsweise einen breiteren Strich in der Mitte machen etc. (Nichts anderes wird gemacht, wenn der Underline-Cursor eingeschaltet wird.) Doch hier erst einmal die vier möglichen Bitkombinationen des Cursormodus:

00 - nicht blinkender Cursor
01 - Cursor ausgeschaltet
10 - langsamer Cursor (Cursor blinkt in 1/16 der BWF)
11 - schneller Cursor (Cursor blinkt in 1/32 der BWF)

BWF = Bildwiederholfrequenz

Wie bereits erwähnt, übernimmt der VDC das Blinken des Cursors inklusive Negieren des Zeichens unter dem Cursor selbständig und entlastet somit die CPU (was zur Geschwindigkeitserhöhung führt).

Beim Blockcursor ist die Startzeile die Zeile 0; die Endzeile - in Register 11 zu definieren - die Zeile 7. Um einen Underline-Cursor zu definieren, braucht man lediglich die Startzeile auf 7 zu ändern.

Um die Auswirkungen zu demonstrieren, probieren Sie einfach folgendes einmal aus:

```
10 REM *** DEMO für Cursor ***
20 A=DEC("D600"): D=DEC("D601")
30 POKE A,10: POKE D,3: REM *** Nicht blinkend - Startzeile=3
40 POKE A,11: POKE D,5: REM *** Endzeile=5
```

Die Cursoradresse wird in den Registern 14 und 15 definiert; an dieser Adresse befindet sich dann der Cursor, der ggf. blinkt und das darunter befindliche Zeichen negiert. Eine andere Funktion haben diese beiden Register nicht.

5.5.5 Die Zeichenlänge und -breite

Die Matrix der im VDC-RAM gespeicherten Zeichen beträgt 8 x 8 Punkte; das heißt nichts anderes, als daß die auf dem Bildschirm dargestellten Zeichen 8 Punkte breit und 8 Zeilen lang sind. Doch dies kann man ändern. In den Registern 22 und 23 kann man die Breite und die Länge der Zeichen bestimmen. Folgendes kleines BASIC-Programm demonstriert dies recht anschaulich:

```
10 REM *** DEMO-Programm für Zeichen-Matrix ***
20 :
30 A=DEC("D600"): D=A+1
40 FOR I0=0 TO 8: POKE A,22: POKE D,112+I0
50 FOR I1=0 TO 8: POKE A,23: POKE D,I1
```

```
60 FOR I2=1 TO 30: NEXT I2,I1
70 FOR I2=1 TO 30: NEXT I2
80 NEXT I0
90 GOTO 40
```

In Register 22 müssen Sie die 112 immer hinzuaddieren, da das obere Nibble unbedingt immer $7 sein muß.

5.5.6 Mehr als 25 Zeilen auf dem Bildschirm

Doch, doch, Sie haben schon ganz richtig gelesen. Es ist möglich, auf Ihrem Bildschirm nicht nur 25 Zeilen mit 2000 Zeichen darzustellen, sondern man kann durch geschickte Manipulation sogar 28 Zeilen mit 2240 (!!) Zeichen und mehr darstellen lassen. Dies ist sicherlich keine Spielerei sondern wird vielmehr jeden engagierten Programmierer erfreuen, der beispielsweise eine Text- oder Dateiverwaltung auf dem C-128 schreiben will; es ist ja nun nicht ganz unbedeutend, ob man nun 2000 Zeichen oder 2240 Zeichen auf dem Bidlschirm zur Verfügung stehen hat.

Die Lösung, die wir Ihnen unterbreiten wollen, beinhaltet die Möglichkeit, weiterhin mit dem BASIC 25 Zeilen zu verwal-ten, d.h. die restlichen 3 Zeilen bleiben beim Scrollen und beim Bildschirm-Löschen unberührt und eignen sich auf diese Weise vorzüglich als Status-Zeilen. Diese 3 Zeilen (inkl. Attribut) können Sie mittels geeigneter Maschinenprogramme verändern. Doch nun zunächst einmal zur Theorie:

Im Register 6 des Video-Controllers kann man angeben, wieviele Zeilen auf dem Bildschirm erscheinen sollen. Hier ist der Normalwert 25. Ändern Sie diesen Wert beispielsweise einmal auf 10:

```
10 A=DEC("D600"): D=A+1
20 POKE A,6: POKE D,10
```

Sie sehen, daß der Controller nun nur noch 10 Zeilen auf dem Bildschirm darstellt, die restlichen Zeilen werden einfach *verschluckt*. So, wie man den Bildschirm kleiner machen kann,

besteht auch die Möglichkeit, die Zeilenanzahl zu erhöhen. Dies tun wir einfach, indem wir die Zeile 20 korrigieren:

```
20 POKE A,6: POKE D,28
```

Und schon, so meint man, müßten die 28 Zeilen auf dem Bildschirm erscheinen. Man erkennt auch einige Zeilen, die zumeist in irgendwelchen schillernden Farben blinken - aber von drei Zeilen kann nicht die Rede sein, man erkennt bestenfalls 2 1/2. Das liegt einfach daran, daß der Bildschirm zu niedrig beginnt. Wir müssen den Bildschirm also nach oben verschieben, um alle Zeilen auch tatsächlich auf dem Bildschirm sehen zu können. Dazu sind die SYNC-Register sehr dienlich - wir wollen das Register 7 - VERTICAL SYNC POSITION - dazu entsprechend verändern. Der Standardwert dieses Registers ist 32. Verändern Sie diesen Wert auf 37, so schiebt sich der Bildschirm langsam nach oben und bleibt dann da auch stehen. Wir können nun (wenn der Monitor entsprechend gut ist) alle 28 Zeilen auf dem Bildschirm erkennen - auch wenn die letzten drei Zeilen noch keinen sinnvollen Inhalt aufweisen.

Noch eine kleine Anmerkung: Auf einem sehr gut eingestellten IBM-Farbmonitor haben wir sogar 30 Zeilen darstellen können. Dies auszunutzen wäre aber unsinnig, da die meisten Monitore dies nicht schaffen würden. Bei zwei oder drei Zusatzzeilen hat aber noch jeder Monitor mitgespielt, so daß man generell sagen kann, daß zumindestens zwei zusätzliche Zeilen möglich sein müßten, die man dann in der Software entsprechend als Status-Zeilen etc. nutzen kann.

Wir wissen bereits, daß das Video-RAM ab Adresse $0000 liegt und das Attribut-RAM ab Adresse $0800. Dies müssen wir korrigieren, da wir nun 2400 Zeichen darzustellen haben; das Ende des Video-RAM läge dann an Adresse $0960 und würde weite Teile des Attribut-RAMs überschreiben und umgekehrt. Zwischen Attribut-RAM und Zeichengenerator ist allerdings genügend Platz - hier brauchen wir uns keinerlei Gedanken zu machen. Es würde sich also die Adresse $0A00 als Startadresse für das Attribut-RAM anbieten.

Wenn wir aber mit BASIC auf dem 80-Zeichen-Schirm schreiben wollen, dann entsteht ein kleines Problem: Der Inter-

preter holt sich die Basisadresse des Attribut-RAMs aus Adresse $0A2F der Zeropage. Nicht weiter schlimm - dann können wir dem BASIC-Interpreter ja die neue Basisadresse mitteilen. Dies ist richtig - doch sieht man sich das Kernal ein wenig genauer an, so stellt man fest, daß die Basisadresse nicht addiert, sondern logisch ODER-verknüpft wird. Dabei werden die Bits 0 und 1 beeinflußt; diese beiden Bits dürfen also nicht relevant sein, d.h. sie dürfen nicht gesetzt sein. Deshalb bietet es sich an, die Adresse $1000 als Basisadresse des Attribut-RAMs zu definieren. Dies tun wir durch die beiden Anweisungen:

```
POKE DEC("0A2F"),16
POKE DEC("D600"),20: POKE DEC("D601"),16
```

Wenn dies getan ist, funktioniert alles wie gehabt. Wir fassen diese Erkenntnisse nun erst einmal in einem Programm zusammen:

```
10 REM *** Demo-Programm für 30-Zeilen-Bildschirm ***
20 :
30 A=DEC("D600"): D=DEC("D601")
40 POKE A,20: POKE D,16: REM *** VDC erhält neue Basisadresse
50 POKE DEC("0A2F"),16:  REM *** Kernal erhält neue Basisadresse
60 POKE A,6: POKE D,38:  REM *** 28 Zeilen anmelden
70 POKE A,7: POKE D,33:  REM *** Neue Synchronisation
```

Wenn Sie dieses Programm starten, so erscheinen 28 Zeilen auf dem Bildschirm - allerdings sind die letzten drei Zeilen noch ohne ernsthaften Inhalt. Diesem Punkt wollen wir uns nun zuwenden.

Beschreiben können Sie diese drei Zeilen durch das PRINT-Kommando leider nicht - hierzu fehlen die notwendigen Voraussetzungen im Betriebssystem. Hieraus wird deutlich, daß wir Zeichen(-ketten) in den Speicher hineinPOKEn müssen. Diese recht aufwendige Arbeit soll Ihnen eine kleine Maschinenroutine abnehmen, so daß die zu druckenden Zeichen doch in einem String zusammengefaßt werden können.

Diese Maschinenroutine wird die Adresse des auszugebenden Strings übergeben. Die Adresse einer Variablen kann man ja mittels dem Kommando *POINTER(Var)* ermitteln. Zuvor

werden noch in den Speicherstellen $FA (250) und $FB (251) das Lo- und das Hi-Byte der Bildschirmadresse übergeben, an der der String dann ausgegeben werden soll. Als Farbe bzw. Attribut wird dann das aktuelle Attribut gesetzt. Beachten Sie aber bitte, daß Sie keine Kontrollzeichen in die Strings integrieren. Diese werden zwar ausgeführt, hinterlassen aber eine Lücke in der Zeichenkette. Ein ordnungsgemäßes Ausführen von Steuersequenzen wäre zwar auch hier möglich - jedoch haben wir aus Platzgründen hierauf zunächst verzichtet; wir wollen ja nicht, daß Sie zum "Meisterabtipper" werden. Die Routine ist also dazu gedacht, Zeichenketten in unserem neuen Fenster auszugeben, ohne daß dadurch größerer Programmieraufwand nötig wird. Um eine beliebige Zeichenkette an die erste Zeile in unserem neuen Fenster auszugeben, wären folgende Kommandos notwendig:

```
T$="Dies ist ein Teststring!"
POKE 250,(2000 AND 255): POKE 251,(2000/256)
A=POINTER(T$)
SYS DEC("D27"),A AND 255,A/256
```

Zuerst wird also die Stringvariable definiert, die die auszugebende Zeichenkette beinhaltet. Dann POKEn wir die Startadresse nach $FA und $FB, zuerst das Lo-, dann das Hi-Byte. Wir ermitteln dann die Adresse, an der der String T$ in der Bank 1 abgelegt ist. Diese Adresse wird dann, wieder in Lo- und in Hi-Byte getrennt, an die Ausgaberoutine an Adresse $0D27 übergeben. Die Routine holt sich dann jedes Zeichen und gibt es aus. Das ist alles. Doch hier zunächst das Maschinenprogramm:

```
00D00  8E 00 D6 STX $D600 ;ABSPEICHERN DES REGISTERS
00D03  2C 00 D6 BIT $D600 ;TESTE STATUS
00D06  10 FB    BPL $0D03 ;NOCH NICHT FERTIG
00D08  8D 01 D6 STA $D601 ;SPEICHERN DES WERTES
00D0B  60       RTS       ;ENDE DER UNTERROUTINE
00D0C  A2 12    LDX #$12  ;UPDATE-REGISTER HI
00D0E  A9 00    LDA #$00  ;LADEN DES HI-WERTES
00D10  20 00 0D JSR $0D00 ;SETZEN DER HI-ADRESSE
00D13  E8       INX       ;UPDATE-ADRESSE LO
00D14  A9 00    LDA #$00  ;LADEN DES LO-BYTES
00D16  20 00 0D JSR $0D00 ;UND AUCH ABSPEICHERN
```

```
00D19  A2 1F     LDX #$1F  ;DATENREGISTER DES VDC
00D1B  A9 00     LDA #$00  ;LADEN DES POKEWERTES
00D1D  20 00 0D  JSR $0D00 ;SETZEN DES WERTES
00D20  A2 12     LDX #$12  ;NOCHMAL DUMMY-BESCHREIBEN
00D22  A9 00     LDA #$00  ;UM UPDATE-ADRESSE AUZULÖSEN
00D24  60        RTS       ;DANN ENDE
00D27  85 FC     STA $FC   ;MERKE LO-BYTE VON STRING
00D29  86 FD     STA $FD   ;MERKE HI-BYTE VON STRING
00D2B  A0 00     LDY #$00  ;OFFSET FÜR LÄNGE STRING
00D2D  A2 01     LDX #$01  ;BANK 1 FUER VARIABLEN
00D2F  A9 FC     LDA #$FC  ;AN ADRESSE $FC STEHT STR.-ADRESSE
00D31  20 74 FF  JSR $FF74 ;UND FAR-FETCH
00D34  85 FE     STA $FE   ;LÄNGE MERKEN
00D36  A0 01     LDY #$01  ;OFFSET FUER LO-BYTE ADRESSE
00D38  A2 01     LDX #$01  ;BANK 1 FUER VARIABLEN
00D3A  A9 FC     LDA #$FC  ;AN $FC BEFINDET SICH DIE ADRESSE
00D3C  20 74 FF  JSR $FF74 ;FAR-FETCH
00D3F  48        PHA       ;RETTE LO-BYTE AUF STACK
00D40  C8        INY       ;ZEIGER AUF HI-BYTE
00D43  A9 FC     LDA #$FC  ;ADRESSE FUER VARIABLE
00D45  20 74 FF  JSR $FF74 ;FAR-FETCH
00D48  85 FD     STA $FD   ;MERKEN DES HI-BYTES
00D4A  68        PLA       ;HOLE LO-BYTE
00D4B  85 FC     STA $FC   ;SPEICHERN DES LO-BYTES
00D4D  A5 FC     LDA $FC   ;HOLE LO-BYTE
00D4F  D0 02     BNE $0D53 ;WENN NICHT NULL, ERNIEDRIGE NUR
00D51  C6 FD     DEC $FD   ;DAS LO-BYTE, SONST ERNIEDRIGE
00D53  C6 FC     DEC $FC   ;AUCH DAS HI-BYTE
00D55  A5 FA     LDA $FA   ;AUCH DIE QUELLADRESSE
00D57  D0 02     BNE $0D5B ;ERNIEDRIGE NUR LO-BYTE
00D59  C6 FB     DEC $FB   ;UM EINS ERNIEDRIGEN, EVTL.
00D5B  C6 FA     DEC $FA   ;AUCH DAS HI-BYTE
00D5D  A5 FA     LDA $FA   ;HOLE LO-BYTE
00D5F  85 E0     STA $E0   ;UND ALS LO-BYTE ZIELADRESSE
00D61  A5 FB     LDA $FB   ;HOLE HI-BYTE
00D63  85 E1     STA $E1   ;UND ALS HI-BYTE ZIELADRESSE
00D65  A2 01     LDX #$01  ;BANK 1 FUER VARIABLEN
00D67  A4 FE     LDY #$FE  ;POSITION IM STRING
00D69  A9 FC     LDA #$FC  ;ADRESSE IN DER ZEROPAGE
00D6B  20 74 FF  JSR $FF74 ;FAR-FETCH
00D6E  A4 FE     LDY $FE   ;HOLE POSITION IM STRING
00D70  84 EC     STY $EC   ;ALS CURSORSPALTE
```

```
00D72  20 0C C0 JSR $C00C ;UND ZEICHEN AUSGEBEN
00D75  C6 FE    DEC $FE   ;ERNIEDRIGE DEN ZEIGER
00D77  D0 E4    BNE $0D5D ;ENDE DES STRINGS NOCH NICHT ERREICHT
00D79  60       RTS       ;ENDE DER UNTERROUTINE
```

Die Routine erscheint Ihnen auf den ersten Blick vielleicht lang, das ist sie aber in Wirklichkeit gar nicht. Bedenken Sie, daß Sie durch diese Routine und einige kleine BASIC-Zeilen drei zusätzliche Zeilen zu Ihrer freien Verwendung haben. Ferner befindet sich am Anfang der Routine noch eine weitere kleine Routine (die wir ja bereits hatten), um an eine beliebige Stelle im VDC-Speicher ein beliebiges Zeichen zu schreiben. (Siehe aber auch Beispiel). Der BASIC-Loader zu dieser Rotuine befindet sich im Anschluß an das Beispielprogramm. Doch hier zunächst das Beispielprogramm, das sowohl 28 Zeilen darstellt, als auch beide neuen Routinen zur Hilfe nimmt.

```
10 REM *** Demo-Programm für 28-Zeilen-Bildschirm ***
20 :
30 A=DEC("D600"): D=DEC("D601")
40 POKE A,20: POKE D,16: REM *** VDC erhält neue Basisadresse
50 POKE DEC("0A2F"),16:  REM *** Kernal erhält neue Basisadresse
60 POKE A,7: POKE D,28:  REM *** 28 Zeilen anmelden
70 POKE A,6: POKE D,33:  REM *** Neue Synchronisation
80 :
90 PRINT CHR$(147);
100 T$="                    ": REM 20 LEERZEICHEN
110 FOR X=0 TO 79 STEP 20: FOR Y=0 TO 2
120 GOSUB 1000: NEXT: NEXT
130 INPUT "Geben Sie Ihren Namen ein: ";T$
140 FOR Y=0 TO 2: X=2*Y: GOSUB 1000: NEXT
150 END
999 :
1000 REM *** AUSGABE VON T$ AN X,Y KOORDINATE; Y=0 BEDEUTET 1.
     ZEILE ***
1010 AZ=2000+Y*80+X: REM ZIELADRESSE
1020 POKE 250,AZ AND 255: REM LO-BYTE
1030 POKE 251,AZ/256: REM HI-BYTE
1040 T%=POINTER(T$): REM ADRESSE DES STRINGS
```

```
1050 SYS DEC("D27"),T% AND 255,T%/256: REM AUSGEBEN
1060 RETURN
1070 :
```

Dieses Programm schaltet also zunächst die zusätzlichen drei Zeilen ein (Zeilen 30-70). Danach wird das Fenster gelöscht, und der von Ihnen eingegebene Name wird in jeder Zeile einmal ausgedruckt.

Wenn Sie das Maschinenprogramm nicht mit dem Direct-Assembler eingeben wollen, so verwenden Sie folgenden BASIC-Loader; Sie können das Maschinenprogramm dann auch als BIN-File auf Diskette sichern.

```
 10 REM *** BASIC-LOADER FÜR PRINTSTRING ***
 20 :
 30 FOR I=DEC("D00") TO DEC("D79")
 40  READ A$
 50  POKE I,DEC(A$)
 60  S=S+DEC(A$)
 70 NEXT
 80 IF S<>13595 THEN PRINT "*** FEHLER IN DATAS ***"
 90 INPUT "SOLL PROGRAMM AUF DISKETTE (J/N): ";A$
100 IF A$<>"J" THEN END
110 INPUT "FILENAME:";F$
120 BSAVE F$,B1,P(DEC("D00")) TO P(DEC("D79")): END
100 :
110 DATA 8E,00,D6,2C,00,D6,10,FB,8D,01,D6,60,A2,12,A9,00
120 DATA 20,00,0D,E8,A9,00,20,00,0D,A2,1F,A9,00,20,00,0D
130 DATA A2,12,A9,00,4C,00,0D,85,FC,86,FD,A0,00,A2,01,A9
140 DATA FC,20,74,FF,85,FE,A0,01,A2,01,A9,FC,A0,74,FF,48
150 DATA C8,A2,01,A9,FC,20,74,FF,85,FD,68,85,FC,A5,FC,D0
160 DATA 02,C6,FD,C6,FC,A5,FA,D0,02,C6,FB,C6,FA,A5,FA,85
170 DATA E0,A5,FB,85,E1,A2,01,A4,FE,A9,FC,20,74,FF,A4,FE
180 DATA 84,EC,20,0C,C0,C6,FE,D0,E4,60
```

5.5.7 Die Hi-Res-GRAFIK

Sicherlich haben wir Sie bereits heiß gemacht, als wir erwähnt haben, daß auch eine Grafikdarstellung auf dem 80-Zeichen-Bildschirm möglich ist. Die Auflösung dieses Grafikmodus

beträgt 640 x 200 Punkte und ist somit also exakt doppelt so hoch wie der Hi-Res-Modus des VIC-Chips. Einen Multi-Color-Modus gibt es hier allerdings nicht. Die Brillanz der Grafik ist allerdings schon beeindruckend (wenn der Monitor mitspielt). Hier braucht man nicht, wie beim VIC manchmal nötig, zwei Punkte nebeneinander einzuschalten, um überhaupt einen Punkt zu erkennen. Allerdings hat man *nur* eine Farbe zur Verfügung, doch für anspruchsvolle Grafiken (beispielsweise mathematische Kurven etc.) vollkommen ausreichend.

Dieser Grafikmodus wird allerdings nicht von den BASIC-7.0-Grafik-Kommandos unterstützt. Wir bieten Ihnen hier wieder ein kleines Maschinensprachepaket an, mit dem Sie folgende elementaren Funktionen ausführen können:

* *Ein- und Ausschalten des Grafikmodus*
* *Löschen der Grafikseite*
* *Setzen und Löschen von Punkten*

Natürlich hätten wir noch mehr Möglichkeiten in das Routinenpaket integrieren können - jedoch soll das *INTERN* ja nicht zur Programmsammlung werden!

Sicherlich ist aber auch beim Grafikmodus des VDC das *WIE* interessant: Durch Setzen des Bit 7 von Register 25 wird der Bit-Map-Mode eingeschaltet. Es sind dann die gesamten 16 KByte des VDC-Speichers für die Grafik auf dem Bildschirm verfügbar. Löscht man dann die Grafik, so wird auch der Zeichengenerator gelöscht. Sollten Sie mittels <RUN/STOP>-<RESTORE> einmal aussteigen, so müssen Sie auch die <ASCII>-<DIN>-Taste betätigen, da Sie sonst wegen des gelöschten Zeichensatzes gar nichts auf Ihrem Bildschirm erkennen. Programmgesteuert kann man natürlich den Zeichensatz nach Umschalten von Grafik- in Textmodus wieder zurückkopieren. Sie können auch während eingeschaltetem Grafikmodus einmal die <ASCII>-<DIN>-Taste betätigen - Sie werden sich wundern.

Durch Setzen des Bit 7 wird also der Grafikmodus eingeschaltet. Da das Attribut-RAM nun aber auch unwirksam wird, es wird ja ebenfalls zur Grafikdarstellung benötigt, müssen wir auch das ATR-Bit im Register 25 löschen. Diese zwei Aktionen können

wir kombinieren, indem wir das Register 25 mit 128 laden. Dies wäre alles, um den Grafikmodus einzuschalten. Attribut-Adresse und Video-RAM-Adresse können wir dort liegen lassen, wo sie standardmäßig definiert sind, dies spielt keinerlei Rolle.

Die Grafik wird ab Adresse $0000 definiert. Die Logik zum Setzen und Löschen eines Punktes ist ähnlich der für den VIC-Chip beschriebenen; hier ist das Setzen und Löschen durch logisches ODER- und UND-Verknüpfen gemeint. Auch beim VDC definiert ein Byte acht Punkte (Pixels). Der erste Punkt, der die Koordinate 0/0 hat, befindet sich in der linken oberen Ecke und somit auch an Adresse $0000. Der weitere Verlauf der Grafik ist einfacher als beim VIC-Chip. Es wird Zeile für Zeile der Grafik definiert. Der Speicherauszug soll durch folgende kleine Grafik verdeutlicht werden:

```
$0000 $0001 $0002 $0003 ..... $027F (639 dezimal)
$0280 $0281 $0282 $0283 ..... $04FF (1279 dezimal)
  .     .     .     .     .     .       .
  .     .     .     .     .     .       .
```

Beim VDC wird der Speicher nicht in Achter-Päckchen aufgeteilt, so daß die Adressierung eines Punktes weitaus einfach realisiert werden kann. Um einen beliebigen Punkt (X/Y) zu adressieren, benötigt man folgende Formel:

AD = INT(X/8) + Y*80

Der Punkt in diesem Byte wird genauso wie beim VIC durch die Formel

2^(7-(X AND 7))

angesprochen. Da diese Adressierung so einfach zu realisieren ist, wurde das Maschinenprogramm auch entsprechend kurz. Zunächst das Assembler-Listing, dann der BASIC-Loader:

```
00C00  4C CD 0C    JMP $0CCD ;Einschalten der Grafik + Löschen
00C03  4C D0 0C    JMP $0CD0 ;Löschen der Grafik
00C06  4C D3 0C    JMP $0CD3 ;Zurück in Textmodus
00C09  4C E0 0C    JMP $0CE0 ;Setzen eines Punktes
00C0C  4C DD 0C    JMP $0CDD ;Löschen eines Punktes
```

```
00C0F  8E 00 D6     STX $D600 ;Register übergeben
00C12  2C 00 D6     BIT $D600 ;Teste Statusbit
00C15  10 FB        BPL $0C12 ;noch nicht fertig
00C17  8D 01 D6     STA $D601 ;Wert übergeben
00C1A  60           RTS       ;Rücksprung aus Unterprogramm
00C1B  8E 00 D6     STX $D600 ;Register übermitteln
00C1E  2C 00 D6     BIT $D600 ;Teste Statusbit
00C21  10 FB        BPL $0C1E ;noch nicht fertig
00C23  AD 01 D6     LDA $D601 ;Hole aktuellen Registerwert
00C26  60           RTS       ;Rücksprung aus Unterprogramm
00C27  A2 19        LDX #$19  ;Register 25 auswählen
00C29  A9 80        LDA #$80  ;Bit 7 setzen - Grafikmodus
00C2B  20 0F 0C     JSR $0C0F ;Register 25 setzen
00C2E  A0 40        LDY #$40  ;$40 Blöcke löschen
00C30  A2 12        LDX #$12  ;Register 18 - Update-Hi
00C32  98           TYA       ;Hi-Byte nach Akku
00C33  20 0F 0C     JSR $0C0F ;Setze Update-Hi
00C36  A2 1F        LDX #$1F  ;Register 31 - DATA-Register
00C38  A9 00        LDA #$00  ;0, da gelöscht wird
00C3A  20 0F 0C     JSR $0C0F ;DATA-Register beschreiben
00C3D  A2 1E        LDX #$1E  ;WORDCOUNT-Register
00C3F  20 0F 0C     JSR $0C0F ;Mit Null belegen
00C42  88           DEY       ;Erniedrige den Zähler
00C43  10 EB        BPL $0C30 ;nächsten Block löschen
00C45  60           RTS       ;Rücksprung aus Löschroutine
00C46  08           PHP       ;Carry: Zeichen für Setzen/Löschen
00C47  A5 FA        LDA $FA   ;Lo-Byte von X-Koordinate
00C49  85 FE        STA $FE   ;zwischenspeichern
00C4B  46 FB        LSR $FB   ;Hi-Byte von X durch zwei
00C4D  66 FA        ROR $FA   ;Carry nach Lo-Byte übertragen
00C4F  46 FB        LSR $FB   ;s.o.
00C51  66 FA        ROR $FA   ;s.o.
00C53  46 FB        LSR $FB   ;ergibt zusammen INT(X/8)
00C55  66 FA        ROR $FA   ;
00C57  A9 00        LDA #$00  ;Hi-Byte der Adresse auf
00C59  85 FD        STA $FD   ;Null setzen
00C5B  A5 FC        LDA $FC   ;Y-Koordinate in Akku merken
00C5D  06 FC        ASL $FC   ;Y mal zwei
00C5F  26 FD        ROL $FD   ;Carry übertragen
00C61  06 FC        ASL $FC   ;nochmal mal zwei ergibt
00C63  26 FD        ROL $FD   ;insgesamt mal 4, plus einmal Y
00C65  65 FC        ADC $FC   ;ergibt Y*5.
```

```
00C67  85 FC      STA $FC    ;Lo-Byte
00C69  90 02      BCC $0C6D  ;Kein Übertrag
00C6B  E6 FD      INC $FD    ;Übertrag nach Hi-Byte
00C6D  A2 04      LDX #$04   ;Es wird jetzt noch 4 mal
00C6F  06 FC      ASL $FC    ;mit zwei multipliziert. Dies
00C71  26 FD      ROL $FD    ;ergibt eine Multiplikation mit 16
00C73  CA         DEX        ;und 16*5 ergibt 80. Y wird also
00C74  D0 F9      BNE $0C6F  ;mit 80 multipliziert.
00C76  A5 FA      LDA $FA    ;INT(X/8)
00C78  65 FC      ADC $FC    ;Addiere zu Y*80
00C7A  85 FC      STA $FC    ;und abspeichern
00C7C  90 02      BCC $0C80  ;Kein Übertrag
00C7E  E6 FD      INC $FD    ;Übertrag berücksichtigen
00C80  A2 12      LDX #$12   ;Register 18 - Update-Hi
00C82  A5 FD      LDA $FD    ;Hi-Byte der errechneten Adresse
00C84  20 0F 0C   JSR $0C0F  ;Wert setzen
00C87  E8         INX        ;Update-Lo
00C88  A5 FC      LDA $FC    ;Lo-Byte der Adresse
00C8A  20 0F 0C   JSR $0C0F  ;Setzen des Lo-Bytes
00C8D  A2 1F      LDX #$1F   ;DATA-Register
00C8F  20 1B 0C   JSR $0C1B  ;Holen des Speicherinhaltes
00C92  48         PHA        ;Rette Wert auf Stack
00C93  A5 FE      LDA $FE    ;Hole X-Koordinate (Lo)
00C95  29 07      AND #$07   ;Nur der Rest X AND 7 ist wichtig
00C97  AA         TAX        ;als Pointer nach X
00C98  68         PLA        ;Hole Speicherwert zurück
00C99  28         PLP        ;Hole Carry zurück
00C9A  B0 05      BCS $0CA1  ;Setzen des Punktes
00C9C  3D C5 0C   AND $0CC5,X ;Löschen des Punktes
00C9F  90 03      BCC $0CA4  ;unbedingter Sprung
00CA1  1D BD 0C   ORA $0CBD,X ;Setzen des Punktes
00CA4  48         PHA        ;Rette neuen Wert
00CA5  A2 12      LDX #$12   ;Update-Hi
00CA7  A5 FD      LDA $FD    ;Hi-Byte von Zieladresse
00CA9  20 0F 0C   JSR $0C0F  ;Setzen des Wertes
00CAC  E8         INX        ;Update-Lo
00CAD  A5 FC      LDA $FC    ;Lo-Byte der Adresse
00CAF  20 0F 0C   JSR $0C0F  ;Setzen des Lo-Bytes
00CB2  A2 1F      LDX #$1F   ;DATA-Register
00CB4  68         PLA        ;Hole Wert wieder von Stack
00CB5  20 0F 0C   JSR $0C0F  ;Setzen des neuen Wertes
00CB8  A2 12      LDX #$12   ;Update-Adresse Hi
```

```
00CBA  4C 1B 0C     JMP $0C1B ;und Punkt setzen
00CBD  80 40 20 10 08 04 02 01 ;Tabelle zum Setzen der Punkte
00CC5  7F BF DF EF F7 FB FD FE ;Tabelle zum Löschen der Punkte
00CCD  20 27 0C     JSR $0C27 ;Setzen des Grafikmodus
00CD0  4C 2E 0C     JMP $0C2E ;Löschen der Grafik
00CD3  A2 19        LDX #$19  ;Register 25 auswählen
00CD5  A9 47        LDA #$47  ;ATR-Bit setzen, TXT-Bit löschen
00CD7  20 0F 0C     JSR $0C0F ;Setzen des Textmodus
00CDA  4C 0C CE     JMP $CE0C ;Kopieren des CHARROM
00CDD  18           CLC       ;Lösche Carry für Punkt löschen
00CDE  90 01        BCC $0CE1 ;unbedingter Sprung
00CE0  38           SEC       ;Setze Carry für Punkt setzen
00CE1  85 FA        STA $FA   ;Abspeichern X-Lo
00CE3  86 FB        STX $FB   ;Abspeichern X-Hi
00CE5  84 FC        STY $FC   ;Abspeichern Y-Koordinate
00CE7  4C 46 0C     JMP $0C46 ;Punkt setzen/löschen
```

Wie Sie sehen, stehen fünf Einsprungadressen zur Verfügung. Beim Einschalten des Grafikmodus wird auch automatisch die Grafikseite gelöscht. Wollen Sie nur die Grafikseite einschalten, so können Sie dies auch einfach mittels folgender BASIC-Kommandos tun:

POKE DEC("D600"),25: POKE DEC("D601"),128

Sie erreichen beim Ansprung der fünf Einsprungadressen folgende Unterroutinen:

$0C00 - Einschalten des Grafikmodus inkl. Löschen
$0C03 - Löschen der Grafik
$0C06 - Zurück in den Textmodus
$0C09 - Setzen eines Punktes
$0C0C - Löschen eines Punktes

Bei den Routinen zum Setzen und Löschen eines Punktes können die Koordinaten direkt beim SYS-Kommando mit übergeben werden. Die Syntax sieht wie folgt aus:

SYS <EINSPRUNG>,<X-LOW>,<X-HIGH>,<Y>

Um also beispielsweise den Punkt mit der Koordinate (185/191) zu setzen, wäre das Kommando

SYS DEC("0C09"),0,185,191

nötig. Genau wie bei der Routine zum Ausdruck eines Strings im Statusfenster sieht auch hier der generelle Aufruf aus:

SYS DEC("0C09"),X AND 255,X/256,Y

Übrigens macht es sich bezahlt, so häufig wie möglich das %-Zeichen an die Variablennamen anzuhängen, da die Variablen dann als Integervariablen behandelt werden - was zu immensen Geschwindigkeitsgewinnen führt. Allerdings geht dies leider nicht bei Schleifenvariablen. Ferner sollte man die Konstanten 255 und 256 als Integervariablen definieren - auch dies bringt Geschwindigkeitsgewinne, da die Werte nicht bei jedem Aufruf neu vom Interpreter berechnet werden müssen. In unserem Beispielprogramm zur Grafik haben wir hiervon Gebrauch gemacht.

Doch hier erst einmal der BASIC-Loader für das Grafikpaket:

```
 10 REM *** BASIC-LOADER FÜR 80-ZEICHEN-GRAFIK ***
 20 :
 30 FOR I=DEC("C00") TO DEC("CE9")
 40 : READ X$: X=DEC(X$)
 50 : POKE I,X
 60 : S=S+X
 70 NEXT
 80 IF S<>25931 THEN PRINT "*** FEHLER IN DATAS ***": END
 90 INPUT "Soll das Programm auf Diskette? (J/N) ";A$
100 IF A$<>"J" THEN END
110 PRINT: INPUT "Filename: ";F$
120 BSAVE F$,B0,P(DEC("C00")) TO P(DEC("CEA"))
130 END
140 :
150 DATA 4C,CD,0C,4C,D0,0C,4C,D3,0C,4C,E0,0C,4C,DD,0C,8E
160 DATA 00,D6,2C,00,D6,10,FB,8D,01,D6,60,8E,00,D6,2C,00
170 DATA D6,10,FB,AD,01,D6,60,A2,19,A9,80,20,0F,0C,A0,40
180 DATA A2,12,98,20,0F,0C,A2,1F,A9,00,20,0F,0C,A2,1E,20
190 DATA 0F,0C,88,10,EB,60,08,A5,FA,85,FE,46,FB,66,FA,46
```

```
200 DATA FB,66,FA,46,FB,66,FA,A9,00,85,FD,A5,FC,06,FC,26
210 DATA FD,06,FC,26,FD,65,FC,85,FC,90,02,E6,FD,A2,04,06
220 DATA FC,26,FD,CA,D0,F9,A5,FA,65,FC,85,FC,90,02,E6,FD
230 DATA A2,12,A5,FD,20,0F,0C,E8,A5,FC,20,0F,0C,A2,1F,20
240 DATA 1B,0C,48,A5,FE,29,07,AA,68,28,B0,05,3D,C5,0C,90
250 DATA 03,1D,BD,0C,48,A2,12,A5,FD,20,0F,0C,E8,A5,FC,20
260 DATA 0F,0C,A2,1F,68,20,0F,0C,A2,12,4C,1B,0C,80,40,20
270 DATA 10,08,04,02,01,7F,BF,DF,EF,F7,FB,FD,FE,20,27,0C
280 DATA 4C,2E,0C,A2,19,A9,47,20,0F,0C,4C,0C,CE,18,90,01
290 DATA 38,85,FA,86,FB,84,FC,4C,46,0C
```

Übrigens befindet sich diese Routine im RS232-Eingabebuffer und kann also von jeder Bankkonfiguration aus aufgerufen werden. Es wurde dieser Speicherbereich ausgewählt, da dieser wohl nur recht selten benötigt wird. Sollten Sie ihn allerdings doch benötigen, so müssen Sie die Routine an eine für Sie geeignete Stelle verschieben und natürlich die nötigen Änderungen am Programm vornehmen.

Zum Schluß wollen wir Sie nicht so allein mit diesen Routinen lassen und haben zu diesem Zweck noch ein kleines Beispielprogramm in BASIC, das Ihnen auf dem 80-Zeichen-Schirm eine gedämpfte Schwingung zeichnet. Wir finden, man kann mit der Ausführungsgeschwindigkeit durchaus zufrieden sein. Sie können dem Beispielprogramm auch die genauere Handhabung der Grafikroutinen entnehmen. Natürlich können Sie auch die zu zeichnende Funktion in Zeile 30 beliebig verändern und sehen, wie "Ihre" Funktion aussieht.

```
10 REM *** BEISPIELPROGRAMM FUER GRAFIKPAKET ***
20 :
30 DEFFNR(X)=40*SIN(X)*EXP(-.05*X)+100
40 FAST: TRAP 220: REM FALLS FEHLER IN FNR(X) AUFTRETEN
50 F%=256: FF%=255: SE=DEC("C09"): RE=DEC("C0C")
60 SYS DEC("C00"): REM GRAFIK EIN
70 Y%=100: REM X-KOORDINATE ZEICHNEN
80 FOR X=0 TO 639 STEP 3
90 : SYS SE,X AND FF%, X/F%, Y%
100 NEXT
110 X%=320: REM Y-KOORDINATE ZEICHNEN
120 FOR Y=0 TO 199 STEP 2
130 : SYS SE,X% AND FF%, X%/F%, Y
```

```
140 NEXT
150 C=-32
160 FOR X=0 TO 639
170 : FU%=FNR(C): IF FU%<0 OR FU%>199 THEN 190
180 : SYS SE,X AND FF%, X/F%, FU%
190 : C=C+.1
200 NEXT
210 GETKEY A$: REM **+ FERTIG, AUF TASTENDRUCK WIEDER TEXT
220 SYS DEC("C06"): PRINT CHR$(147)
```

Sicherlich gibt es für die Grafik eine Flut an Anwendungsmöglichkeiten. Wir wollen Ihrer Phantasie hier freien Lauf lassen und wünschen Ihnen viel Erfolg beim Gebrauch der Grafikroutinen.

6. Das Memory-Management - Die MMU

6.1 Einführung die MMU

Die sogenannte *MEMORY MANAGEMENT UNIT (MMU)* wurde hergestellt, um die im C128 anfallenden komplexen Aufgaben der Adressierung zu lösen. Wie Sie vielleicht wissen, kann sowohl die 8502 als auch der Z-80 lediglich 64 KByte auf einmal adressieren. Sie wissen sicherlich schon von BASIC her, daß man die beiden verschiedenen RAM-Banks nur getrennt ansprechen kann. Das liegt daran, daß je 64 KByte RAM, das ROM und die I/O-Bausteine überlagert werden mußten. Beispielsweise gibt es an der Adresse $D600 sowohl zweimal RAM als auch I/O (der 80-Zeichen-Controller) und zusätzlich noch ROM. Werden Cartridges in den Expansion-Slot eingeführt, so muß hier noch weiter differenziert werden.

Die MMU wird auch im 64er-Modus benutzt - Sie ist voll kompatibel zum C64. Zusätzlich kann sie dann noch die Aufgaben erfüllen, die im C128 und CP/M-Modus anfallen. Sie nimmt die Rechnermodus-Auswahl vor und ihr obliegt auch die Auswahl, ob der Z-80 oder die 8502 arbeitet. Hier eine Liste ihrer Features:

* *Generierung und Verwaltung des angepaßten Adreßbusses (Translated-Address-Bus TA8-TA15)*
* *Auswahl des Rechnermodus (C64, C128, CP/M)*
* *Auswahl des arbeitenden Prozessors (Z-80B, 8502)*
* *Bereitstellung und Verwaltung der CAS-Auswahl-Leitungen für die Speicherbankumschaltung des RAM.*

Die MMU hat insgesamt 11 Register, die sich ab der I/O-Adresse $D500 befinden. Da der I/O-Bereich nun nicht immer eingeschaltet ist, werden das Konfigurationsregister sowie die Laderegister A bis D in den Speicherbereich $FF00 bis $FF05 kopiert. Dadurch kann man vier fest gespeicherte Konfigurationen, die sich in den Präkonfigurationsregistern A bis D befinden, durch einfaches Beschreiben eines Laderegisters in das Konfigurationsregister laden, ohne den I/O-Bereich einstellen zu müssen. Dies ist eine sehr nützliche Einrichtung und erspart

einiges an Zeit und Programmieraufwand. Doch hierzu später mehr.

Hier zunächst eine grafische Darstellung der verfügbaren Register:

$FF04	LCRD	Load Configuration Register D
$FF03	LCRC	Load Configuration Register C
$FF02	LCRB	Load Configuration Register B
$FF01	LCRA	Load Configuration Register A
$FF00	CR	CONFIGURATION REGISTER (Kopie von $D501)

$D50B	VR	Version-Register
$D50A	P1H	Page-1-Pointer-Hi
$D509	P1L	Page-1-Pointer-Lo
$D508	P0H	Page-0-Pointer-Hi
$D507	P0L	Page-0-Pointer-Lo
$D506	RCR	RAM Configuration Register
$D505	MCR	Mode Configuration Register
$D504	PCRD	Preconfiguration Register D
$D503	PCRC	Preconfiguration Register C
$D502	PCRB	Preconfiguration Register B
$D501	PCRA	Preconfiguration Register A
$D500	CR	CONFIGURATION REGISTER (Kopie an $FF00)

6.2 Das Konfigurationsregister

Wie bereits erwähnt, befindet sich ab Adresse $FF00 (unabhängig von der eingeschalteten RAM-Bank) eine Kopie einiger MMU-Register. Dies ist eigentlich nicht ganz korrekt ausgedrückt: In Wirklichkeit befindet sich an $FF00 lediglich die Kopie eines Registers; es handelt sich hier um das Konfigurationsregister CR. Liest man die Speicherstelle $FF00 aus, so erhält man den aktuellen Wert des Konfigurationsregisters. Beschreibt man die Adresse $FF00, so ändert sich auch entsprechend augenblicklich das Konfigurationsregister an Adresse $D500 in der MMU. Die Register $FF01 bis $FF04 sind eigentlich bloß "halbe" Kopien der MMU-Register. Halb deshalb, weil man zwar beim Auslesen den aktuellen Wert des korrespondierenden Präkonfigurationsregisters erhält, beim Beschreiben eines dieser Register allerdings nicht die korrespondierenden Register in der MMU ändert, sondern den Inhalt des Konfigurationsregisters.

Dies ist aber kein Nachteil, ganz im Gegenteil. Beschreibt man ein LCRx-Register, so wird das CR (bitte erlauben Sie uns hier die in der Tabelle angegebenen Abkürzungen, weil sonst das Buch wegen der langen Bezeichnungen unnötig in die Länge gezogen wird) mit dem korrespondierenden PCR geladen. Kleines Beispiel: Wir beschreiben das LCRA an Adresse $FF01. Der Inhalt dieses Registers ändert sich dabei nicht, wohl aber der Inhalt des CRs. Es wird nämlich das PCRA ($D501) in das CR kopiert. Dies ist eine sehr praktische Einrichtung: Ohne den System-I/O-Bereich einblenden zu müssen, können wir das CR verändern. Dabei kann man zwischen vier in der MMU zwischengespeicherten Konfigurationen auswählen. Das heißt, der Programmierer braucht bloß zu sagen: "Wähle Konfiguration #1", und die MMU schaltet diese Konfiguration ein. In Maschinensprache sähe die Auswahl einfach so aus:

```
STA $FF01 ;Akku-Inhalt ist gleichgültig - Konfig. 1 einschalten
```

Am Anfang eines Programms könnte man die vier PCRs mit den meistbenutzten Konfigurationen vorprogrammieren. Aber auch eine "manuelle" Konfigurationsänderung ist nicht viel komplizierter. Man lädt den Akku mit dem für die Konfigura-

tion notwendigen Bitmuster und speichert diesen an Adresse $FF00 ab. Beispiel für BANK 15:

```
LDA #00   ;entspricht BANK-15-Kommando
STA $FF00 ;Konfiguration auswählen
```

Beim Konfigurationsregister sind alle 8 Bits relevant:

Bit 7,6 Auswahl der anzusprechenden RAM-Bank. Möglich sind in der 128-K-Version die Bitkombinationen 00 und 01. Da aber eine Erweiterung bis zu 256 KBytes RAM vorgesehen ist, existieren noch die Möglichkeiten 10 und 11 für diese Erweiterung. Sind diese RAM-Banks noch nicht vorhanden, so ist 10 gleichbedeutend mit 00 und 11 mit 01.

Bit 5,4 Auswahl, was bei der Adressierung des Speicherbereichs $C000 bis $FFFF angesprochen werden soll:
00 - System ROM (Kernal)
01 - Internal Function ROM
10 - External Function ROM
11 - RAM (Bank, siehe Bits 6 und 7)

Bit 2,3 Auswahl, was bei der Adressierung des Speicherbereichs $8000 bis $BFFF angesprochen werden soll:
00 - System ROM (BASIC)
01 - Internal Function ROM
10 - External Function ROM
11 - RAM (Bank, siehe Bits 6 und 7)

Bit 1 Auswahl, was bei der Adressierung des Speicherbereichs $4000 bis $7FFF angesprochen werden soll:
0 - System ROM (BASIC)
1 - RAM (Bank, siehe Bits 6 und 7)

Bit 0 Auswahl, ob bei der Adressierung des Speicherbereichs $D000 bis $DFFF System-I/O oder RAM/ROM angesprochen werden soll:
0 - System-I/O
1 - RAM/ROM, abhängig von Bits 4 und 5

Zu beachten ist, daß bei eingeschaltetem ROM im Bereich $C000 bis $FFFF (Bits 4 und 5) immer eine Lücke im Bereich $D000 bis $DFFF existiert. Ist das System-I/O eingeschaltet, so belegen die System-I/O-Bausteine diesen Bereich. Ist Bit 0 aber gesetzt, so befindet sich hier der Zeichengenerator.

6.2.1 Die Präkonfigurationsregister

Die Präkonfigurationregister befinden sich an den Adressen $D501 bis $D504, die Kopien der Register an den Adressen $FF01 bis $FF04. Sie sind im C64-Modus ohne Bedeutung. Präkonfigurationregister sind Register in der MMU, die auf bestimmte Speicherkonfigurationen vorprogrammiert werden können. Diese vorprogrammierten Konfigurationen kann man dann mittels der "Load Configuration Register" ins Konfigurationsregister übertragen. Wie man sich diese Register zu Nutze macht, ist im vorangehenden Kapitel 6.2 beschrieben. Die Bits sind ebenso aufgeschlüsselt wie beim Konfigurationsregister. Auch die Aufschlüsselung finden Sie im Kapitel 6.2 vor.

6.3 Das Mode-Configuration-Register

Das Mode-Configuration-Register befindet sich an Adresse $D505. Es bestimmt den aktuellen Rechnermodus, also welche CPU arbeitet (8502 oder Z-80 B) und ob der 64er- oder der 128er-Modus aktiv ist.

Bit 7 — **Zeigt an, ob beim RESET die 40/80-Zeichen-Taste gedrückt war. 0=80-Zeichen-, 1=40-Zeichen-Modus.**

Bit 6 — **In diesem Bit wird angezeigt, ob der 128er- oder der 64er-Modus aktiv ist; 0=128er-Modus. Nach dem Einschalten oder dem RESET ist immer zunächst der 128er-Modus aktiv.**

Bits 4,5 — **Diese beiden bidirektionalen Bits zeigen an, ob die Steckmodulleitungen GAME- oder EXROMIN belegt sind. Ist dies der Fall, so muß der 64er-Modus eingeschaltet und die Kontrolle an die Cartridge übergeben werden. Im 128er-Modus werden diese Leitungen nicht belegt.**

Bit 3 FSDIR-Kontrollbit. Dieses Bit wird als Ausgabebit für den schnellen seriellen Datenbus-Buffer sowie als Eingabebit für das Disk-Enable-Signal benutzt.

Bits 1,2 Diese Bits sind (noch) ohne Bedeutung.

Bit 0 In diesem Bit wird die Prozessorauswahl vorgenommen; 0=Z80, 1=8502.

Wird das Bit 0 des Registers beschrieben, so wird dieses zwischengespeichert, bis ein Taktzyklus zu Ende ist. Es könnte sonst wegen der Prozessorumschaltung zu Komplikationen kommen.

Mittels des Bit 0 können wir also bestimmen, ob die Z-80 oder die 8502 arbeiten soll. Das Bit wird beim Beschreiben des Registers bis zum Abfallen des Taktimpulses zwischengespeichert, um Komplikationen zu vermeiden. Ist das Bit gesetzt, also die Z-80 aktiv, so wird der Bereich $D000 bis $DFFF in den Bereich $0000 bis $0FFF gespiegelt. Hier ist auch das BIOS-ROM physikalisch zu finden. Das BIOS-ROM ist bei eingeschalteter 8502 unter gar keinen Umständen (per Software) auszulesen.

Wird beispielsweise die Z-80 eingeschaltet, so wird die 8502 gestoppt und die Z-80 macht da weiter, wo sie aufgehört hat. Das heißt nichts anderes, als daß an PC (*Program Counter*) mit dem Programmablauf fortgefahren wird. Genauso verhält sich dies natürlich beim Einschalten der 8502: Sie nimmt ihre Arbeit da auf, wo sie aufgehört hat; dies ist unmittelbar nach dem Kommando zum Einschalten der Z-80.

Im 64er-Modus wird die Z-80-Enable-Leitung (die durch Bit 0 definiert wird) immer auf Null gehalten, damit man die Z-80 im 64er-Modus nicht einschalten kann. Ferner befindet sich im 64er-Modus an den Adressen ab $FF00 keine Kopie der MMU-Register mehr.

6.4 Das RAM-Configuration-Register

Das RAM-Konfigurationsregister befindet sich an Adresse $D506 der MMU. Es dient dazu, die gemeinsamen RAM-Bereiche zu definieren. Wozu aber überhaupt gemeinsame RAM-Bereiche definieren?

Ganz einfach: Der Interpreter beispielsweise hat ja bekanntlich in der Zero-Page (die mittlerweile eigentlich schon gar keine Zero-Page im herkömlichen Sinne mehr ist) alle notwendigen Systemvariablen und -pointer abgespeichert. Schaltet der Interpreter beispielsweise jetzt auf RAM-Bank 1, um Variablen zu lesen oder zu beschreiben, so wären diese Systemzeiger nicht mehr verfügbar - da sie sich in der RAM-Bank 0 befinden. Es wäre recht umständlich, wenn man bei jeder Änderung einer Zero-Page-Adresse diese Änderung in beiden RAM-Banks vornehmen müßte.

Um dieses hin und her zu vermeiden, hat man sich gedacht, es wäre doch ganz praktisch, wenn man einen bestimmten Speicherbereich so definieren könnte, daß er in allen RAM-Banks gleich aussieht. In Wirklichkeit ist natürlich nur in einer RAM-Bank die Zero-Page gespeichert, und zwar in RAM-Bank 0. Spricht man nun in RAM-Bank 1 (oder einer anderen) diesen Speicherbereich an, so erkennt die MMU dies und adressiert entsprechend die RAM-Bank 0.

Diesen gemeinsamen Speicherbereich nennt man *Common Areas*. Die MMU bietet dem Programmierer hier aber sogar die Möglichkeit zu definieren, ob er eine Common Area wünscht, und wenn ja, wie groß diese sein soll und wo sie liegen soll. Doch zunächst die Registerbelegung, bevor wir auf die einzelnen Bits näher eingehen wollen:

Bits 6,7 In diesen beiden Bits wird die RAM-Bank für den VIC-Chip bestimmt, also wo die Text- oder Grafikseite abgespeichert ist. Normalerweise befindet sich das Video-RAM in RAM-Bank 0.
00=Ram-Bank 0,
01=RAM-Bank 1,
10=RAM-Bank 2(0),
11=RAM-Bank 3(1)

Bits4,5 Diese beiden Bits sind in der vorliegenden Version noch ohne Bedeutung. Sie sind für eine evtl. Erweiterung auf 1 MByte RAM vorgesehen. Dann selektieren diese beiden Bits den anzusprechenden 256-KByte-Block.

Bits 2,3 Bits 2 und 3 zeigen an, wo ein gemeinsamer Bereich definiert ist:
00=Kein gemeinsamer Bereich, unabhängig von Bits 0 und 1
01=Unterer Bereich ist gemeinsam
10=Oberer Bereich ist gemeinsam
11=Sowohl unterer als auch oberer Bereich sind gemeinsam

Bits 0,1 Hier wird definiert, wieviel KBytes als Common Area verwendet werden sollen. Diese beiden Bits haben nur Gültigkeit, wenn Bits 2 und 3 ungleich 00 sind.
00=1 Kbyte gemeinsamer Bereich
01=4 KByte gemeinsamer Bereich
10=8 KByte gemeinsamer Bereich
11=16 KByte gemeinsamer Bereich

Wenn man eine Common Area definiert, so ist der kleinstmögliche Bereich 1 KByte. Doch ist es auch möglich, gar keinen Bereich als gemeinsam zu deklarieren. Dazu muß man Bit 2 und 3 auf Null setzen. Ist nur eines der beiden Bits 0 und 1 gesetzt, haben die Bits 4 und 5 Wirkung. Normalerweise ist nur der untere Bereich mit einem KByte als Common Area definiert. Im 64er-Modus hat dieses Register keinerlei Wirkung.

Hat man ein KByte als Common Area definiert, so ist der Bereich $0000 bis $03FF in beiden RAM-Banks identisch, wenn der untere Bereich eingeschaltet worden ist. Schaltet man sowohl den unteren als auch den oberen Bereich als Common Area ein, so ist der Bereich $0000 bis $03FF und $FC00 bis $FFFF in beiden RAM-Banks identisch. Man kann also bis zu 32 KByte als Common Area definieren, indem man beide Bereiche und 16 KByte als Common Area definiert.

Übrigens wird physikalisch beim Abspeichern und Auslesen der Common Area immer auf RAM-Bank 0 zugegriffen.

Die Bits 6 und 7 bestimmen, aus welcher RAM-Bank sich der VIC-Chip seine Informationen bezüglich des Video-RAM holen soll. An dieser Stelle bietet sich eine fantastische Möglichkeit: Man kann sehr einfach zwei 40-Zeichen-Bildschirme verwalten, ohne die Adresse des Video-RAM verschieben zu müssen, was

etwas komplizierter ist, als das Umschalten der RAM-Bank. In RAM-Bank 0 kann man ab Adresse $0400 Bildschirm Nr.1 definieren und in RAM-Bank 1 an derselben Adresse den Bildschirm Nr. 2. Umschalten kann man dann, indem man Bit 6 setzt oder löscht, das ist schon alles.

```
LDA #00    ;System-I/O
STA $FF00 ;einschalten
LDA $D506 ;alter RCR-Wert
ORA #$40   ;Bildschirm in RAM-Bank 1
STA $D506 ;einschalten
```

6.5 Die Seitenzeiger

Higher Byte ($D508/$D50A)

Bits 7-4	unbenutzt
Bits 3-0	Adreßbits 16 bis 19 für Translates Address (TA) In der vorliegenden Version ist nur das Bit 0 interessant, die restlichen Bits 1-3 werden ignoriert.

Lower Byte ($D507/$D509)

Bits 0-7	Diese Bits stellen das Higher Byte des Seitenzeigers dar, also die Adreßbits 8-15. Beim Zeropage-Zeiger ist dieses Byte standardmäßig auf 0, beim Page-1-Zeiger auf 1.

Es gibt gleich zwei Seitenzeiger: einen Seitenzeiger für die Zeropage und einen Seitenzeiger für Page 1, in der normalerweise der System-Stack liegt.

$D507/$D508: Page-Pointer 0
$D509/$D50A: Page-Pointer 1

Das Lowbyte dieser Zeiger stellt die Adreßbits 8 bis 15 dar. Das Highbyte bestimmt die RAM-Bank, die dafür genutzt werden soll, stellt also die Adreßbits 16 bis 19 dar. Die Bits 7 bis 4 werden in den Highbytes nicht belegt.

Wird das Highbyte eines Seitenzeigers beschrieben, so wird dies so lange zwischengespeichert, bis auch das Lowbyte des Zeiger beschrieben worden ist.

Wenn die Zeropage oder die Page eins an eine andere Adresse verlegt wird, so addiert die MMU bei Zugriffen auf die Zeropage bzw. bei Stapelaktionen automatisch die Basisadresse.

Die Vorteile liegen klar auf der Hand: Man kann sich beispielsweise für jede Unterroutine einen eigenen Stapel anlegen sowie einen eigenen Systemvariablenbereich, wenn man nicht die Kernal-Routinen aufruft. Was die Zeropageverschiebung betrifft, so hat dies zwei Vorteile:

Die Programme werden kürzer und schneller

Es kommt ja nun nicht gerade selten vor, daß der Assembler-Programmierer in der Zeropage nach freien Speicherstellen sucht. Beispielsweise funktioniert das LDA ($xx),Y-Kommando bekanntlich nur mit Zeropage-Adressen. Hierzu kann man dann die Zeropage in geeignete Speicherbereiche verschieben.

Praktisch ist auch die Möglichkeit, die Page 1 zu verschieben. Dadurch kann man sich bei jeder Unterroutine einen eigenen Stapel anlegen. Dies ist eine sehr nützliche Einrichtung. Man braucht lediglich den Stapelzeiger zu retten und kann dann über einen neuen eigenen Stapel für die Unterroutine verfügen. Man hat dadurch

a) mehr Platz auf dem Stapel und braucht
b) den Stapel nicht wieder vollkommen zu rekonstruieren,

wenn man die Routine verläßt. Man muß lediglich den Page-1-Pointer wieder auf Normalwert ($01) und den Stapelzeiger SP zurückzusetzen. Besonders für PASCAL-Compiler ist dies eine sehr nützliche Sache.

Eine Verschiebung des Stapels könnte beispielsweise so aussehen:

```
LDA #$00        ;System-I/O
STA $FF00       ;einschalten
LDA #$F0        ;Stapel ab Adresse $F000
STA $D507       ;in RAM-Bank 0
TSX             ;und SP retten
STX $FD         ;in Zeropage $FD
LDX #$FF        ;neuer Stack wird
TXS             ;initialisiert
```

Am Ende der Routine, die den Stapel umdefiniert hat, muß man den Stack wieder rekonstruieren, da sonst kein Rücksprung aus dem Unterprogramm mittels RTS mehr möglich ist. Diese Rekonstruktion sieht dann entsprechend so aus:

```
LDX $FD         ;Hole alten Stackpointer
TXS             ;und SP rücksetzen
LDA #$01        ;Stack wieder an Adresse $0100
STA $D507       ;also "Normalwert"
RTS             ;Rücksprung jetzt möglich
```

Wir wollen Ihnen hier eine etwas unkonventionelle Art des Bildschirmlöschens vorstellen: Sie ist bei wirklich professionellen Programmierern sehr beliebt, weil sie sehr schnell ist. Man verwendet sie beispielsweise bei Grafikprogrammen auch zum Ausfüllen von Flächen.

Das ganze sieht so aus, daß wir den Stackpointer zum Adressieren praktisch "mißbrauchen". Durch ein PHA wird der Inhalt des Akkus an die aktuelle Stapeladresse geschrieben und der Stapelzeiger automatisch erniedrigt - dies alles in einem Ein-Byte-Befehl und viel schneller, da dies alles hardwaremäßig geschieht. In "normaler" Assembler-Schreibweise sähe dies so aus:

```
STA ($xx),Y
DEY
```

Die Adressierungsart ist komplizierter für die CPU - braucht also mehr Zeit. Dieselbe Aktion benötigt drei Byte - ist wiederum langsamer, da der Code geholt, erkannt und ausgeführt werden muß.

Unsere neue Löschroutine rettet den Stack-Pointer, legt ihn dann auf den Bildschirmstart $0400 um und pusht dann 1024 mal den Akku in den neuen Stack. Natürlich muß man nach jeweils 256 Bytes das Hi-Byte entsprechend inkrementieren. Zur Sicherheit sollte man während der Aktion den Interrupt sperren, damit es nicht zu einem Stapelüberlauf kommt.

```
     LDA #$00  ;Bank 15
     STA $FF00
     SEI       ;Interrupts sperren
     LDA #$04  ;Neue Startadresse des SP
     STA $D509 ;ist $0400 in RAM-Bank 0
     TSX       ;Stack-Pointer nach X
     STX $FD   ;und aktuellen Zeiger retten
     LDX #$FF  ;SP auf Anfang des Stacks
     TXS       ;legen
     LDY #$03  ;Viermal 256 Bytes
     LDX #$00  ;256-Byte-Zähler
     LDA #$20  ;Füllzeichen
NEXT PHA       ;Zeichen speichern
     DEX       ;Schleife erniedrigen
     BNE NEXT  ;Nächstes Zeichenn
     INC $D509 ;Erhöhe Hi-Byte des SP
     DEY       ;Alle vier Blöcke gefüllt?
     BNE NEXT  ;Nein, noch nicht
     LDX $FD   ;alten SP holen
     TXS       ;und wieder merken
     LDA #$01  ;Hi-Byte des Originalstacks
     STA $D509 ;und setzen
     CLI       ;Interrupts wieder zulassen
     RTS       ;Ende der Löschroutine
```

Ich glaube nicht, daß diese Routine viel länger als eine "herkömmliche" Löschroutine ist. Sie ist aber in jedem Fall schneller. Außerdem demonstriert sie sehr schön die Möglichkeiten, die sich durch die Manipulationsmöglichkeit der Page-Pointer-Basisadressen eröffnen.

6.6 Das Version Register

Bits 7-4 Bank Version; Diese Bits geben Auskunft über den total verfügbaren Speicherplatz. Wie bereits erwähnt, ist der Rechner für Ein Mega-Byte konzipiert. Standardbelegung dieses Registers beim 128er ist $20. Die "2" steht hier wohl für zwei 64-KByte-Blöcke. Eine Ein-Mega-Byte-Version würde also sechszehn 64-KByte-Blöcke beinhalten. Bits 7-4 dieses Registers enthielten dann eine 0.

Bits 3-0 MMU Version; Es wird die Versionsnummer der MMU angegeben.

Das System-Versionsregister ist für die eigentliche Speicherverwaltung recht uninteressant. Im niederwertigen Halbbyte befindet sich eine Angabe für vorhandene MMU-Version. Im höherwertigen Halbbyte ist der vorhandene Speicherausbau nachzulesen. Hier können Programme feststellen, auf wieviel Speicher zurückgegriffen werden kann und sich darauf einstellen. Zukünftige Programme werden dies sicherlich auch tun (müssen).

7. Die Assembler-Programmierung

7.1 Einführung in die Assembler-Programmierung

Was Assembler bzw. Maschinensprache ist, brauchen wir Ihnen als INTERN-Leser wohl kaum zu erklären. Dieses Kapitel soll Ihnen vielmehr erläutern, wie man die Betriebssystem-Routinen sinnvoll in eigene Programme einbinden kann. Wozu das Rad immer wieder neu erfinden? Es stehen schließlich eine Reihe nützlicher Routinen zur Verfügung, auf die man wirklich leicht zurückgreifen kann. Ihre Programme werden dadurch kürzer und leichter zu lesen.

Wir wollen Ihnen also die Arbeit erleichtern und Ihnen die nützlichsten Kernal-Routinen anhand kleiner Beispiele erläutern. Natürlich können wir dabei unmöglich auf alle Kernal-Routinen eingehen, es sind einfach zu viele.

7.2 Die CPU - die 8502

Herz des Rechners ist und bleibt die CPU und dies ist im Commodore 128, neben der Z-80B, die 8502. Sie ist softwaremäßig voll kompatibel zur 6510 und deren Vorgänger 6502. Im Gegensatz zur 6510 ist sie allerdings im Stande, mit 2 MHz getaktet zu werden - sie ist also doppelt so schnell.

Die Pinbelegung:

1 0IN Systemtakt Eingang; wahlweise 1 oder 2 MHz (ca.)

2 RDY Ready;
0=Prozessor hält beim nächsten Lesezyklus an, bis RDY=1. Von dieser Möglichkeit macht man beispielsweise beim Betrieb langsamer Speicher Gebrauch.

3 -IRQ Interrupt Request;
0=Prozessor holt sich die nächste Befehlsadresse von $FFFE und macht dort weiter. Dieser Umstand tritt nur ein, wenn der Interrupt erlaubt war.

4 -NMI Non-Maskable Interrupt;
0= Prozessor holt sich die nächste Befehlsadresse von $FFFA und macht dort weiter. Diesen Interrupt kann man nicht verhindern: deshalb auch Non-Maskable.

5 AEC Address Enable Control;
0= Prozessor bringt Daten-, Adreß- und Steuerbus in den hochohmigen Zustand (Tri-State). Der Bus kann nun von anderen Einheiten betrieben werden, beispielsweise ein zweiter Prozessor.
6 VCC Betriebsspannung +5V
7-20 A0-A13; Adreßbus
21 GND
22-23 A14-A15; Adreßbus
24-29 P5-P0; I/O-Pins
30-37 D7-D0; Datenbus
38 R/-W; 0=Schreibzugriff, 1=Lesezugriff
Alle Zugriffe finden nur während 02=1 statt.
39 02OUT; Systemtakt Ausgang zur Versorgung anderer Bausteine
40 RES Reset; 0=Prozessor geht in den Ruhezustand. Beim Übergang von 0 nach 1 holt sich der Prozessor eine Adresse von $FFFC und beginnt dort mit dem Programm.

7.3 Die Kernal-Routinen und wie man sie nutzt

Zuerst wollen wir uns den Routinen widmen, die sich teilweise in der erweiterten Zeropage befinden. Es handelt sich hierbei um die sehr wichtigen Routinen, die es einem ermöglichen, eine beliebige Speicherstelle in jeder beliebigen Bank zu lesen, zu beschreiben oder einen Vergleich damit durchzuführen.

7.3.1 FETCH, STASH und CMPARE

Diese drei Routinen dienen also dem Laden, dem Speichern und dem Vergleichen einer Speicherstelle in einer beliebigen Bank, unabhängig von der aktuellen Konfiguration. Nach dem Aufruf einer dieser Routinen ist die Konfiguration auch weiterhin unverändert.

Beim Aufruf dieser Routinen muß man im X-Register den Konfigurationsindex übergeben. Das Betriebssystem hat 16 Konfigurationen der 128 möglichen in einer Tabelle ab $F7F0 gespeichert, hieraus wird dann die entsprechende Konfiguration geholt und im Konfigurationsregister abgelegt.

X-Reg	Speicherkonfiguration
0	nur RAM 0
1	nur RAM 1
2	nur RAM 2 (RAM 0)
3	nur RAM 3 (RAM 1)
4	Int. ROM, RAM 0, I/O
5	Int. ROM, RAM 1, I/O
6	Int. ROM, RAM 2, I/O
7	Int. ROM, RAM 3, I/O
8	Ext. ROM, RAM 0, I/O
9	Ext. ROM, RAM 1, I/O
10	Ext. ROM, RAM 2, I/O
11	Ext. ROM, RAM 3, I/O
12	Kernal, Int (Lo), RAM 0, I/O
13	Kernal, Ext (Lo), RAM 1, I/O
14	Kernal, BASIC, RAM 0, CHARROM
15	Kernal, BASIC, RAM 0, I/O

Entnehmen Sie die gewünschte Speicherkonfiguration dieser Tabelle und laden Sie das X-Register damit.

7.3.1.1 FETCH

Ein Teil der FETCH-Routine befindet sich an Adresse $02A2 im RAM. Will man eine Speicherstelle auslesen, so sind folgende Übergabeparameter notwendig:

Akku: Zeiger auf Zeropage-Adresse
X-Reg: Konfigurationsindex (s.o.)
Y-Reg: Offset für Adresse

Bevor man diese Routine anspringt, muß man also an irgendeiner Adresse in der Zeropage die Adresse der Speicherstelle, die man laden will, ablegen. Diese Zeropage-Adresse wiederum muß man der Routine im Akku übergeben. Folgendes Beispielprogramm würde die Adresse $1000 aus RAM-Bank 1 auslesen:

```
LDA #$00  ;Lo-Byte von $1000
STA $FC   ;in Zeropage
LDA #$10  ;Hi-Byte von $1000
```

```
STA $FD   ;in Zeropage
LDA #$FC  ;Adresse in Zeropage
LDY #$00  ;Offset ist null
LDX #$01  ;RAM-Bank 1 ist auszuwählen
JSR $FF74 ;FETCH - Übergabe in Akku
```

Nach dem Aufruf der Routine befindet sich im Akku der Inhalt der ausgelesenen Speicheradresse. Das X-Register beinhaltet die aktuelle Konfiguration und das Y-Register bleibt unverändert.

7.3.1.2 STASH

Die STASH-Routine ist praktisch das Gegenstück zur FETCH-Routine. Allerdings ist die Anwendung dieser Routine nicht vollkommen identisch. Da man im Akku den zu speichernden Wert übergeben muß, kann man nicht mehr im Akku die Adresse des Zeropage-Pointers übergeben. Deswegen muß man das, was einem die FETCH-Routine noch abgenommen hat, *zu Fuß* erledigen. Ein Beschreiben der Speicheradresse $1000 in RAM-Bank 1 sieht beispielsweise wie folgt aus:

```
LDA #$00  ;Lo-Byte von $1000
STA $FC   ;in Zeropage
LDA #$10  ;Hi-Byte von $1000
STA $FD   ;in Zeropage
LDA #$FC  ;Zeropage-Adresse des Zeigers
STA $02B9 ;in STASH-Routine schreiben
LDA #$FF  ;Wert, der geschrieben wird
LDX #$01  ;RAM-Bank 1
LDY #$00  ;Offset für Adresse
JSR $FF77 ;STASH ausführen
```

Nach Aufruf der STASH-Routine ist der Akku und das Y-Register unverändert, das X-Register beinhaltet die aktuelle Konfiguration.

Auf dieselbe Art und Weise könnte man beispielsweise die MMU-Register beschreiben, ohne extra die Konfiguration ändern zu müssen. Dies gilt natürlich für alle anderen I/O-Bausteine gleichermaßen.

7.3.1.3 CMPARE

Die CMPARE-Routine vergleicht eine beliebige Speicherstelle mit dem Inhalt des Akkus. Dazu muß man, wie bei der STASH-Routine, die Adresse der zu vergleichenden Speicherstelle zunächst in die CMPARE-Routine ins RAM schreiben. Ein Vergleichen der Speicherstelle $1000 in RAM-Bank 1000 mit dem Wert $22 sieht wie folgt aus:

```
LDA #$00  ;Lo-Byte von $1000
STA $FC   ;in Zeropage
LDA #$10  ;Hi-Byte von $1000
STA $FD   ;in Zeropage
LDA #$FC  ;Adresse des Pointers in der Zeropage
STA $02C8 ;der CMPARE-Routine die Adresse mitteilen
LDA #$22  ;Wert, mit dem verglichen wird
LDX #$01  ;RAM-Bank 1
LDY #$00  ;Offset
JSR $FF7A ;Vergleichen von ($1000) in RAM 1 mit $22
```

Nachdem die Routine aufgerufen wurde, sind die Flags (Zero, Minus und Carry) entsprechend dem Vergleich gesetzt. Der Akku und das Y-Register bleiben unverändert, das X-Register beinhaltet die aktuelle Speicherkonfiguration.

7.3.2 GETCONF

Diese Routine macht nichts anderes, als das Konfigurationsbyte entsprechend dem Konfigurationsindex im X-Register aus der Tabelle ab $F7F0 zu holen. Dieser Wert wird lediglich übergeben, nicht aber gesetzt. Da man das Kernal-ROM sowieso eingeschaltet haben muß, um diese Routine anspringen zu können, empfiehlt es sich, das Konfigurationsbyte selbst aus der Tabelle auszulesen; es geht einfach schneller.

```
LDX #$0F  ;Konfiguration 15 auswählen
JSR $FF6B ;GETCONF
STA $FF00 ;Konfiguration setzen
```

wäre also gleichbedeutend mit

```
LDX #$0F    ;Konfiguration 15 auswählen
LDA $F7F0,X ;Konfigurationsbyte holen
STA $FF00   ;Konfiguration setzen
```

Der Speicheraufwand ist derselbe - man kann ihn sogar verkürzen, indem man direkt (ohne X-Register) sagt:

```
LDA $F7F0+$0F
```

7.3.3 JSRFAR und JMPFAR

Hat man beispielsweise Teile des ROMs ausgeblendet und will trotzdem eine Kernal-Routine anspringen, so kann man dies über die Routine JSRFAR tun. Hierbei dienen allerdings nicht mehr die CPU-Register als Parameterübergabe sondern die Zeropage-Adressen $02 bis $09.

$02	**Konfigurationsindex**
$03, $04	**Neuer PC - also Sprungadresse**
$05	**Neuer Prozessorstatus**
$06	**Akku**
$07	**X-Register**
$08	**Y-Register**
$09	**SP - Stackpointer**

Als Ausgangsparameter befinden sich ab $05 die entsprechenden Werte. Nehmen wir einmal an, wir haben Konfiguration 1 eingeschaltet - also nur RAM 1. Wir wollen nun den Inhalt der Adresse $0400 in RAM-Bank 0, also der linken oberen Ecke des 40-Zeichen-Bildschirmes, erfahren. Dazu müssen wir uns der FETCH-Routine bedienen. Dies sieht dann so aus:

```
LDA #$4F  ;RAM 1 & Kernal einschalten
STA $FF00 ;ins Configuration Register
```

```
LDA #$0F  ;Konfigurationsindex Kernal & RAM 0
STA $02   ;übergeben
LDA #$FF  ;Hi-Byte von $FF74
STA $03   ;übergeben
LDA #$74  ;Lo-Byte der Zieladresse
STA $04   ;$FF74 übergeben
LDA #$00  ;Lo-Byte von $0400
STA $FC   ;merken
LDA #$04  ;Hi-Byte von $0400
STA $FD   ;übergeben
LDA #$FC  ;Zeropage-Adresse des Pointers
STA $06   ;und übergeben
LDA #$00  ;RAM-Bank 0 adressieren
STA $07   ;Wert für X-Register für FETCH
LDA #$00  ;Wert für Y-Register für FETCH
STA $08   ;Offset speichern
JSR $FF6E ;JSRFAR aufrufen
LDA $06   ;Hier steht der Wert von $0400 in RAM 0
```

Eine Menge Parameter, die übergeben werden müssen. Es ist zunächst sehr wichtig, drauf zu achten, daß der Kernal-Bereich $C000 bis $FFFF eingeschaltet ist. Hier darf also kein RAM adressiert werden, die JSRFAR-Routine würde sich aufhängen, selbst wenn Sie die JSRFAR-Routine direkt an $02CD aufrufen würden - es wird nämlich zurück ins Kernal verzweigt. Darum sollten Sie vor dem JSRFAR-Aufruf unbedingt das beachten, was auch in unserer Beispielroutine als aller erstes getan wird: Durch das Byte $7F wird die RAM-Bank 1 eingeschaltet, und alle Speicherbereiche außer $C000 bis $FFFF werden auf RAM geschaltet. Dann wird das neue Konfigurationsregister definiert.

Der zweite sehr wichtige Punkt ist: Der Programm-Counter PC wird mit Hi-Byte an Adresse $03 und Lo-Byte an Adresse $04 definiert; es ist also nicht die übliche Assembler-Schreibweise.

Alle Angaben, die nicht unbedingt notwendig sind, können weggelassen werden. Meist reicht es aus, lediglich die Speicherkonfiguration an $02 und den neuen Program Counter PC an $03/$04 zu definieren. Alles andere sind Optionen, die einem dann und wann vielleicht mal dienlich sein können.

Die Routine JSRFAR beschreibt auf jeden Fall die Adressen $05 bis $09 beim Rücksprung mit den entsprechenden Werten, so daß sie gegebenenfalls hier zur Verfügung stehen. In unserer Beispielroutine wird ja auch von der Parameterübergabe im Akku Gebrauch gemacht.

Wir wollen nun noch ein weiters kleines Beispiel aus der Praxis geben. Nehmen Sie einmal an, Sie hätten sowohl in RAM-Bank 0 als auch in RAM-Bank 1 Programmcode. Hier zunächst die *Unterroutine* in RAM-Bank 1, die in unserem Beispiel nichts anderes machen soll, als Akku und X-Register zu addieren. Der Übertrag wird durch das Carry-Flag angezeigt. Geben Sie die Routine im Monitor durch *A 12000* ein. Sie wählen dann RAM-Bank 1.

```
12000        LDA $06   ;Akku-Parameter
12002        CLC       ;Lösche Carry für Addition
12003        ADC $07   ;Addiere X-Register
12005        RTS       ;Ende der Routine
```

Die Routine holt sich nun aus Adresse $06 den Inhalt des übergebenen Akkus und addiert dann das X-Register. Damit ist die Routine praktisch schon zu Ende. Der Inhalt des Akkus wird an Adresse $06 übergeben. In diesem Beispiel ist es wichtig, daß an Adresse $05 der Prozessorstatus, also die Flags übergeben werden. Im Hauptprogramm kann man dann nach dem Aufruf der Routine mit BCC oder BCS das Carry-Flag abprüfen. Doch hier die Routine in RAM-Bank 0, die die Routine in RAM-Bank 1 mittels der JSRFAR-Routine aufruft:

```
02000        LDA #$3F  ;RAM 0 und Kernal
02002        STA $FF00 ;als Konfiguration setzen
02005        LDA #$0D  ;RAM 1 und Kernal
02007        STA $02   ;neue Konfiguration
02009        LDA #$20  ;Akku ist $20
0200B        STA $06   ;übergeben
0200D        LDA #$FF  ;Es soll $FF addiert werden
0200F        STA $07   ;übergeben
02011        LDA #$20  ;Hi-Byte von $2000
02013        STA $03   ;als PC übergeben
02015        LDA #$00  ;Lo-Byte von $2000
02017        STA $04   ;als PC übergeben
```

```
02019        JSR $FF74 ;JSRFAR aufrufen
0201C        LDA $05   ;Flags holen
0201E        PHA       ;auf Stack und
0201F        PLP       ;ins Flag-Register
02020        LDA $06   ;Lo-Byte der Addition
02022        STA $FD   ;als Lo-Byte speichern
02024        LDA #$00  ;Hi-Byte
02026        STA $FE   ;speichern
02028        BCC $202C ;Kein Übertrag - dann Sprung
0202A        INC $FE   ;Übertrag berücksichtigen
0202C        BRK       ;in den Monitor
```

Wenn Sie diese Routine eingegeben und gestartet haben, werden Sie an Adresse $FD/$FE das Ergebnis der Addition $FF + $20 = $11F vorfinden. Sie können anhand dieser Routine erkennen, wie man die Flags, die an Speicherstelle $05 übergeben werden, auch tatsächlich in das Status-Register der CPU bekommt: Durch PUSHen des Akkus; man braucht dann diesen Wert nur noch ins Status-Flag zu POPpen.

Die JMPFAR-Routine funktioniert genauso wie die JSRFAR-Routine. Allerdings gibt es hier keinen Rücksprung mittels RTS, aber deswegen heißt diese Routine ja auch JMPFAR. Natürlich können auch keine Ausgabeparameter abgefragt werden, da es keinen Rücksprung gibt.

7.4 Die wichtigsten Kernal-Routinen

7.4.1 Kernal-Routinen mit Vektoren ab $FF4D

Wir wollen uns zunächst mit den Kernal-Routinen beschäftigen, die durch Sprungvektoren ab Adresse $FF4D definiert werden. Es handelt sich hierbei um die wohl wichtigsten Routinen, die von der Ein-/Ausgabe eines Zeichens bis hin zu den RS-232-Routinen reichen.

Die Routinen werden in der Reihenfolge Ihrer Definition ab $FF4D aufgeführt. Soweit möglich, werden die Ein-/Ausgabeparameter angegeben, auch eine kurze Beschreibung und wenn es geboten erscheint ein kleines Beispielprogramm für

diese Routine. Die Einsprungadressen der Routinen sind hexadezimal und dezimal (in Klammern) angegeben.

Wenn Vektoren vorhanden sind, sollten Sie immer nur über diese springen, Vektoren sind schließlich dazu da. Sollte sich das Betriebssystem einmal ändern oder sollten Ergänzungen gemacht werden, so sind die Vektoren nicht davon betroffen und gewährleisten, daß Ihr Programm nicht "abstürzt" oder verrückt spielt. Wenn Sie dies beachten, so haben Sie mit Sicherheit weniger Arbeit bei der Programmierung.

C64MODE

Zweck: Einschalten des 64er-Modus
Adresse: $FF4D (65357)

Beschreibung: Wird diese Routine angesprungen, so wird vom 128er- in den 64er-Modus umgeschaltet. Es wird die Taktfrequenz auf 1 MHz reduziert und die MMU verriegelt alle nötigen Register, damit man aus dem 64er-Modus heraus nicht mehr manipulieren kann. Es gibt keine Rückkehr!

Eingabeparameter: keine
Ausgabeparameter: -entfällt, da keine Rückkehr-

DMA-CALL

Zweck: Initialisiert externe RAM-Bausteine
Adresse: $FF50 (65360)

Beschreibung: Um einen direkten Speicherzugriff (DMA=Direct Memory Access) auf das externe RAM zu haben, muß dieses erst durch diese Routine initialisiert werden. Im X-Register wird die neu einzuschaltende Konfiguration übergeben.

Eingabeparameter: .X
Ausgabeparameter: entfällt

BOOTCALL

Zweck: Es wird die Diskette gebootet.
Adresse: $FF53 (65363)

Beschreibung: Beim Aufruf dieser Routine wird versucht, die eingelegte Diskette zu booten - es passiert also dasselbe wie beim Einschalten des Gerätes. Kann die Routine kein Boot-File finden, so gibt Sie die Kontrolle zurück. Im X-Register wird die Geräteadresse angegeben, ob also von Floppy 8 oder 9 gebootet werden soll.

Eingabeparameter: .X
Ausgabeparameter: entfällt

PHOENIX

Zweck: Kaltstart
Adresse: $FF56 (65366)

Beschreibung: Kaltstart des 128er-Modus. Sollte sich eine Speicherkarte in der Expansion-Cartridge befinden, so wird die Kontrolle an diese Karte übergeben. Ansonsten wird auch hier versucht, die Diskette zu booten. Tabulatoren, KEY-Definitionen etc. werden rückgesetzt.

LKUPLA

Zweck: Sucht in Tabelle nach logischer Filenummer
Adresse: $FF59 (65369)

Beschreibung: Die Routine sucht in der Tabelle der Geräte- und Sekundäradressen nach der in <Akku> übergebenen logischen Filenummer. Die Statusvariable ST wird entsprechend dem Verlauf der Routine gesetzt. Wird die logische Filenummer gefunden, so ist das CARRY gelöscht und es werden folgende

Parameter übergeben: A=LFN, X=Geräteadresse, Y=Sekundäradresse. Bei Mißerfolg wird das CARRY gesetzt. Es können nur logische Filenummern gefunden werden, die auch mit OPEN geöffnet wurden!

Eingabeparameter: .A enthält zu suchende LFN
Ausgabeparameter: Status ST an $90, .A, .X, .Y, CARRY
Zeropage-Adressen $B8 bis $BA.

Beispiel:

```
;Suchen einer LFN
LDA #$01  ;Es soll nach LFN
JSR $FF59 ;1 gesucht werden.
BCS ERROR ;Noch kein OPEN erfolgt---Fehler ausgeben
TAX       ;LFN nach X
JSR $FFC9 ;CKOUT - Datei als Ausgabedatei setzen
```

LKUPSA

Zweck: Suchen einer Sekundäradresse
Adresse: $FF5C (65372)

Beschreibung: Es wird in der Tabelle der geöffneten Kanäle nach der im Y-Register übergebenen Sekundäradresse gesucht. Genauso wie bei der Routine LKUPLA wird auch hier das Carry bei Mißerfolg der Suche gesetzt. Bei Erfolg wird das Carry gelöscht, der <Akku> enthält die LFN, das X-Register die Geräteadresse und das Y-Register die Sekundäradresse.

Eingabeparameter: .Y enthält zu suchende SA
Ausgabeparameter: Status ST an $90, .A, .X, .Y, CARRY
Zeropage-Adressen $B8 bis $BA.

Beispiel:

```
;Suchen der LFN des Floppy-Befehlskanals
LDY #$0F   ;Suche LFN mit
JSR $FF5C  ;Sekundäradresse 15
BCS ERROR  ;Nicht gefunden, dann Fehler melden
TAX        ;LFN nach X
JSR CKOUT  ;und als Ausgabegerät öffnen
JSR INITD  ;Diskette initialisieren
```

SWAPPER

Zweck: Umschalten von 40/80-Zeichen
Adresse: $FF5F (65375)

Beschreibung: Diese Routine tauscht den 40/80-Zeichen-Modus aus. Dabei müssen die Informationen in der Zeropage für den aktiven Bildschirm mit dem passiven Bildschirm vertauscht werden. Es wird der Speicherbereich $E0 bis $FA mit dem Bereich $0A40 bis $0A5A ausgetauscht. Es sind keine Eingangsparameter notwendig.

Beispiel:

```
;Löschen beider Bildschirme
JSR $C142 ;Bildschirm löschen
JSR $FF5F ;40/80-Zeichen-Modus tauschen
JSR $C142 ;passiven Bildschirm auch löschen
JSR $C142 ;wieder auf aktuellen Bildschirm zurück
```

DLCHR

Zweck: Kopieren des CHARROM
Adresse: $FF62 (65378)

Beschreibung: Beim Betätigen der ASCII/DIN-Taste wird der Zeichensatz erneut ins VDC-RAM kopiert, da der 80-Zeichen-Controller sich die Informationen zur Zeichendarstellung nicht aus dem ROM holt. Von dieser Routine wird beispielsweise in dem Grafik-Paket Gebrauch gemacht, da bei der Grafik der Zeichensatz im VDC-RAM überschrieben wird. Mit dieser

Routine wird der mit der ASCII/DIN-Taste ausgewählte Zeichensatz ins VDC-RAM kopiert. Es gibt weder Ein- noch Ausgabeparameter.

PFKEY

Zweck: Umdefinieren eines KEYs
Adresse: $FF65 (65381)

Beschreibung: Mit dieser Routine sind Sie in der Lage, die Funktionstasten beliebig zu belegen. (F1 bis F10 sowie SHIFT/RUN-STOP und HELP). Im <Akku> wird die Adresse in der Zeropage angegeben, die als Pointer auf den KEY-Text zeigt. Ferner übergeben Sie im X-Register die Nummer der Funktionstaste (1 bis 12) und im Y-Register die Länge des Strings. Dann können Sie die Routine PFKEY aufrufen, die diesen String in die Tabelle einfügt.

Eingabeparameter: Zeropage, .A, .X, .Y

Beispiel: (An Adresse $2100)

```
;Help-Taste umdefinieren
LDA #$00  ;Lo-Byte von $2000
STA $FC   ;in Zeropage merken
LDA #$20  ;Hi-Byte von $2000
STA $FD   ;in Zeropage merken
LDA #$FC  ;Pointer
LDX #$0C  ;Help-Taste umbelegen
LDY #$04  ;Anzahl
JSR $FF65 ;Taste umdefinieren
```

Und an Adresse $2000:

```
02000 52 55 4E 0D .....
```

SETBNK

Zweck: Speicherbank für Disk-Operation definieren
Adresse: $FF68 (65384)

Beschreibung: Diese Routine muß vor LOAD, SAVE, VERIFY und jedem OPEN-Kommando aufgerufen werden. Es werden der Konfigurationsindex des Filenamens im Y-Register sowie der Konfigurationsindex des zu bearbeitenden Speicherbereichs im <Akku> übergeben. Das Y-Register wird in der Zeropage-Adresse $C6, der <Akku> an $C7 gespeichert. Siehe auch Beispiel bei SETNAM (FFBD).

Eingabeparameter: .A, .Y

GETCONF

Zweck: Holen des Konfigurationsbytes
Adresse: $FF6B (65387)

Beschreibung: Wie Sie ja bereits wissen, existiert eine Tabelle von 16 für den Normalbedarf ausreichenden Konfigurationsbytes. Diese Tabelle befindet sich an Adresse $F7F0. Sie übergeben der Routine im X-Register den Konfigurationsindex und erhalten im Akku das Konfigurationsbyte, das man normalerweise dann ins Konfigurationsregister an Adresse $FF00 der MMU schreibt.

Eingabeparameter: .X
Ausgabeparameter: .A

Beispiel:

```
;Setzen der RAM-Bank 1
LDX #$01  ;nur RAM-Bank 1
JSR $FF6B ;Hole Konfig.-Byte
STA $FF00 ;und setzen
```

JSRFAR

Zweck: Sprung in Unterroutine in beliebiger Bank
Adresse: $FF6E (65390)

Beschreibung: Die Routine JSRFAR dient dazu, ein Unterprogramm in einer beliebigen Konfiguration anzuspringen. Die Parameterübergabe findet an den Zeropage-Adresssen $02 bis

$09 statt. Nach Abschluß der Routine wird die alte Konfiguration wieder hergestellt. Eine sehr genaue Beschreibung inkl. Beispielprogramm finden Sie in Kapitel 7.3.3.

Eingabeparameter: Zeropage $02 bis $09
Ausgabeparameter: Zeropage $05 bis $09

JMPFAR

Zweck: Sprung in beliebige Bank
Adresse: $FF71 (65393)

Beschreibung: Auch hier findet die Parameterübergabe an den Zeropage-Adressen $02 bis $09 statt. Allerdings ist JMPFAR kein Unterprogrammaufruf sondern lediglich ein Sprung in eine beliebige Bank; JMPFAR faßt also das Umschalten des Konfigurationsbytes und den Sprung zusammen. Da es hier keine Rückkehr gibt, werden auch keine Parameter zurückgegeben.

Eingabeparameter: Zeropage $02 bis $09

INDFET

Zweck: Holen eines Bytes aus beliebiger Bank
Adresse: $FF74 (65396)

Beschreibung: Diese Routine, die sich übrigens hauptsächlich in der Zeropage befindet, ermöglicht Ihnen, jede beliebige Speicheradresse in jeder beliebigen Konfiguration auszulesen, ohne daß Sie Ihre aktuelle Konfiguration merklich verändern müßten. Dazu definieren Sie zuerst in einer Zeropage-Adresse den Zeiger auf den auszulesenden Speicherplatz. Im <Akku> wird dann diese Zeropage-Adresse übergeben, im X-Register der Konfigurationsindex und im Y-Register der Offset zum Zeropage-Zeiger.

Eingabeparameter: .A, .X, .Y, 1 Zeropage-Adresse
Ausgabeparameter: .A

Beispiel:

```
;Hole $1000 aus RAM-Bank 1
LDA #$00  ;Lo-Byte von $1000
STA $FC   ;in Zeropage merken
LDA #$10  ;Hi-Byte von $1000
STA $FD   ;in Zeropage merken
LDA #$FC  ;Zeiger in Zeropage
LDX #$0D  ;RAM 1 und Kernal
LDY #$00  ;Offset ist Null
JSR $FF74 ;Hole Byte aus $1000, RAM-Bank 1
```

INDSTA

Zweck: Akku in beliebiger Bank abspeichern
Adresse: $FF77 (65399)

Beschreibung: Ähnlich wie die INDFET-Routine speichert diese Routine den Akku-Inhalt in einer beliebigen Speicherkonfiguration ab. Dazu müssen ebenfalls die Parameter in <Akku>, X- und Y-Register übergeben werden. Allerdings wird im Akku das abzuspeichernde Zeichen übergeben! Die Zeropage-Adresse, an der der Zeiger gespeichert ist, müssen Sie selbst an Adresse $02B9 definieren.

Eingabeparameter: .A, .X, .Y, Zeropage, $02B9

Beispiel:

```
;Abspeichern von $FF an $1000 in RAM-Bank 1
LDA #$00  ;Lo-Byte von $1000
STA $FC   ;merken
LDA #$10  ;Hi-Byte von $1000
STA $FD   ;auch merken
LDA #$FC  ;Adresse in Zeropage
STA $02B9 ;INDSTA-Routine mitteilen
LDA #$FF  ;Wert, der geschrieben werden soll
LDX #$0D  ;RAM 1 und Kernal
LDY #$00  ;Offset ist Null
JSR $FF77 ;INDSTA aufrufen
```

INDCMP

Zweck: Vergleich <Akku> mit <Speicher> bel. Bank
Adresse: $FF7A (65402)

Beschreibung: Diese Routine vergleicht den <Akku> mit einer beliebigen Speicherstelle in einer beliebigen Bank. Genauso wie bei der INDSTA-Routine müssen Sie die Adresse des Zeigers in der Zeropage der INDCMP-Routine mitteilen. Dies tun Sie an Adresse $02C8. Im <Akku> wird das zu vergleichende Zeichen übergeben, im X-Register der Konfigurationsindex und im Y-Register der Offset. Nach dem Aufruf der Routine befindet sich das Ergebnis des Vergleiches - also das Statusbyte des Prozessors - an Adresse $05. Entnehmen Sie dem Beispiel, wie Sie dann entsprechend auf den Vergleich reagieren können.

Eingabeparameter: .A, .X, .Y, Zeropage, $02C8
Ausgabeparameter: $05 (Status)

Beispiel:

```
;Vergleiche <Akku> mit $1000 in RAM-Bank 1
LDA #$00  ;Lo-Byte von $1000
STA $FC   ;merken
LDA #$10  ;Hi-Byte von $1000
STA $FD   ;auch merken
LDA #$FC  ;Zeiger in Zeropage
STA $02C8 ;INDCMP-Routine mitteilen
LDA #$FF  ;Vergleichsoperand
LDX #$0D  ;RAM-Bank 1 und Kernal
LDY #$00  ;Offset
JSR $FF7A ;INDCMP aufrufen
LDA $05   ;Status holen (Ergebnis des Vergleichs)
PHA       ;auf Stack um dann ins
PLP       ;Statusbyte des Prozesors zu holen
BEQ EQUAL ;Wenn gleich, dann Sprung
;--- Ungleich ---
```

PRIMM

Zweck: Ausgabe eines Textes
Adresse: $FF7D (65402)

Beschreibung: Diese Routine ist sehr praktisch, da einfach anzuwenden. Es brauchen keinerlei Parameter übergeben zu werden. Alle Zeichen, die dem Aufruf folgen, werden auf das aktuelle Ausgabegerät über BSOUT ausgegeben. Als Endezeichen wird ein Nullbyte verwendet. Mit der Ausführung des Programmes wird dann an der Stelle fortgefahren, die dem Nullbyte unmittelbar folgt. Einziger Nachteil dieser Routine: Beim Disassemblieren wird das Programm unübersichtlich.

Beispiel:

```
;Beispiel für PRIMM-Routine
JSR $FF7D ;Nachfolgende Zeichen ausgeben
.ASC "Dies ist ein Text !!!"
.BYT $0D,$0A,$0D,$00
LDA #$00  ;Hier wird mit dem Programm fortgefahren
```

Siehe auch Beispiel im ROM-Listing an $F908

CINIT

Zweck: Video-Controller und Editor initialisieren
Adresse: $FF81 (65409)

Beschreibung: Es werden die Funktionstasten auf Standard gelegt, beide Video-Controller initialisiert und der 40/80-Zeichen-Modus in Abhängigkeit der 40/80-Zeichen-Taste eingeschaltet. Weiterhin werden der Tastaturbuffer gelöscht sowie alle Flags rückgesetzt und ein CLRCH ausgeführt.

IOINIT

Zweck: Initialisierung der Ein/Ausgabegeräte
Adresse: $FF84 (65412)

Beschreibung: Die Ein/Ausgabegeräte werden initialisiert, d.h. die RESET-Leitung auf dem IEC-Bus wird aktiviert. Angeschlossene Drucker werden in den Anfangszustand versetzt und die Floppy löscht ihre Kanäle - hört sich also an, als ob sie gerade eingeschaltet worden wäre.

RAMTAS

Zweck: BASIC-Warmstart
Adresse: $FF87 (65415)

Beschreibung: Diese Routine initialisiert die Zeropage, setzt die Zeiger für SYSTOP und SYSBOT (also die Speicherunter- und -obergrenze), setzt die Zeiger für die RS-232-Ein/Ausgabebuffer und den Kassettenbuffer zurück.

RESTOR

Zweck: Systemvektoren initialisieren
Adresse :$FF8A (65418)

Beschreibung: Es werden die Systemvektoren ab Adresse $0314 bis $0332 (inkl.) auf Normalwert gesetzt. Diese Routine sollte aufgerufen werden, wenn Sie zu viele Vektoren verbogen und die Übersicht verloren haben oder wenn Sie beispielsweise ein Erweiterungspaket ausschalten wollen. Diese Routine ruft die folgende VECTOR-Routine mit gelöschtem CARRY auf.

VECTOR

Zweck: Systemvektoren kopieren oder rücksetzen
Adresse: $FF8D (65421)

Beschreibung: Diese Routine kopiert die 16 Vektoren ab $0314 in den durch das X- (Low) und Y-Register (High) definierten Speicher, sofern das CARRY-Flag gesetzt ist. Bei gelöschtem CARRY-Flag werden die Vektoren ab $0314 mit dem durch das X- und Y-Register angegebenen Bereich geladen.

Eingabeparameter: .X, .Y, CARRY

Beispiel:

```
LDX #$00  ;Lo-Byte von $1000
LDY #$10  ;Hi-Byte von $1000
CLC       ;Lösche Carry zum Kopieren ($1000)->($0314)
JSR $FF8D ;Belege Vektoren neu
```

SETMSG

Zweck: DOS-Meldungen ermöglichen/verhindern
Adresse: $FF90 (65424)

Beschreibung: Die Routine speichert den Wert des <Akku> in der Zeropage-Adresse $9D. Sollen Systemmeldungen ausgegeben werden, so ist das Bit 7 des <Akkus> zu setzen. Ist $9D positiv, so werden Systemmeldungen verhindert.

Eingabeparameter: .A

SECND

Zweck: Sekundäradresse nach LISTEN senden
Adresse: $FF93 (65427)

Beschreibung: Es wird die zu sendende Sekundäradresse im <Akku> übergeben. Die Routine gibt den <Akku> dann als Sekundäradresse auf dem IEC-Bus aus.

Eingabeparameter: .A

Beispiel:

```
;LISTEN wurde gesendet
LDA #$F0  ;Sekadr. 0 bei CLOSE
JSR $FF93 ;Sekundäradresse senden
```

TKSA

Zweck: Sekundäradresse nach TALK senden
Adresse: $FF96 (65430)

Beschreibung: Adäquat zu der vorhergehenden Routine sendet diese Routine die Sekundäradresse - übergeben im <Akku> - nach erfolgtem TALK-Signal an den IEC-Bus.

Eingabeparameter: .A

MEMTOP

Zweck: Setzen/Holen der Speicherobergrenze
Adresse: $FF99 (65433)

Beschreibung: Ist das CARRY-Flag gesetzt, so wird im X-Register (Lo) und Y-Register (Hi) die maximal verfügbare Speicherstelle übergeben. Wird die Routine mit gelöschtem CARRY angesprungen, so wird die Speicherobergrenze mit den beiden Registern belegt.

Eingabeparameter: .X, .Y (bei gelöschtem CARRY), CARRY
Ausgabeparameter: .X, .Y (bei gesetztem CARRY)

Beispiel:

```
;Auslesen der Speicherobergrenze
SEC       ;Auslesen der Obergrenze
JSR $FF99 ;Hole Obergrenze
STX $FC   ;zwischenspeichern
STY $FD   ;zwischenspeichern
LDX #$00  ;Lo-Byte von $1000
LDY #$10  ;Hi-Byte von $1000
CLC       ;Flag zum Setzen des MEMTOP
JSR $FF99 ;Setze Speicherobergrenze
```

MEMBOT

Zweck: Setzen/Holen der Speicheruntergrenze
Adresse: $FF9C (65436)

Beschreibung: Genauso wie bei der Routine MEMTOP wird bei gelöschtem CARRY-Flag die Untergrenze des verfügbaren Speichers mit den beiden Registern X (Lo) und Y (Hi) belegt. Ist das CARRY-Flag gesetzt, so wird die Speicheruntergrenze ausgelesen und in den beiden Registern übergeben.

Eingabeparameter: .X, .Y (bei gelöschtem CARRY), CARRY
Ausgabeparameter: .X, .Y (bei gesetztem CARRY)

KEY

Zweck :Ermitteln gedrückter Tasten
Adresse: $FF9F (65439)

Beschreibung: Diese Routine ist elementar zur Tastaturdekodierung. Die Tastatur wird auf eine gedrückte Taste anhand der Tastaturdekodiertabellen überprüft. Wird eine gedrückte Taste ermittelt, so wird der ASCII-Wert errechnet und dieser dem Tastaturbuffer (ab $034A) hinzugefügt.

SETTMO

Zweck: Setzen des Timeout-Flags für IEEE
Adresse: $FFA2 (65442)

Beschreibung: Die Routine speichert den im <Akku> übergebenen Wert als Timeout-Flag für die IEEE-Routinen an Adresse $0A0E. Um den Timeout in den IEEE-Routinen zu ermöglichen, muß das Bit 7 des <Akkus> gesetzt sein.

Eingabeparameter: .A

ACPTR

Zweck: Holt ein Byte vom seriellen Bus
Adresse: $FFA5 (65445)

Beschreibung: Die Routine holt ein Byte vom seriellen IEC-Bus. Das geholte Zeichen wird im Akku übergeben. Das Statusbyte ST an $90 wird entsprechend der Aktion gesetzt.

Ausgabeparameter: .A

CIOUT

Zweck: Ausgabe eines Zeichens auf IEC-Bus
Adresse: $FFA8 (65448)

Beschreibung: Diese Routine ist das Gegenstück zu ACPTR. Das im <Akku> übergebene Zeichen wird auf dem IEC-Bus ausgegeben. Auch hier wird das Statusbyte ST an $90 entsprechend der Aktion geändert.

Eingabeparameter: .A

UNTLK

Zweck: UNTALK auf IEC-Bus senden
Adresse: $FFAB (65451)

Beschreibung: Diese Routine wird beim Schließen bzw. Umlegen eines Eingabekanals aufgerufen. Sie bringt das zum Reden (TALK) gebrachte Gerät zum Schweigen.

UNLSN

Zweck: UNLISTEN auf IEC-Bus senden
Adresse: $FFAE (65454)

Beschreibung: Entsprechend zu UNTALK wird bei dieser Routine ein empfangendes Gerät vorerst abgeschaltet. Dies wird beim Schließen oder Umlegen eines Ausgabekanals gemacht.

LISTN

Zweck: Senden von LISTEN an ein Gerät
Adresse: $FFB1 (65457)

Beschreibung: Es wird ein am IEC-Bus angeschlossenes Gerät zum Empfang aufgefordert. Dazu wird das Signal LISTEN über den IEC-Bus geschickt. Im <Akku> wird die Geräteadresse des anzusprechenden Gerätes übergeben. Beispielsweise wird bei einem Drucker ein LISTEN gesendet, bevor die Zeichen zur Ausgabe über den IEC-Bus wandern. Wenn Sie LISTEN verwenden, so müssen Sie die auszugebenden Zeichen über die Routine CIOUT ausgeben (nicht über BSOUT!!). Zum Schließen des Kanals verwenden Sie dann die Routine UNLISTEN. Es kann immer nur ein Gerät am IEC-Bus aktiv sein. Um diese komplizierten Arbeiten zu vereinfachen, können Sie im Betriebssystem Kanäle öffnen und schließen. BSOUT und BASIN übernehmen dann das Senden von LISTEN und UNLISTEN sowie TALK und UNTALK.

Eingabeparameter: .A

Beispiel:

```
;LISTEN an Drucker senden
LDA #$24  ;Geräteadresse Drucker & LISTEN ein
JSR $FFB1
```

TALK

Zweck: Senden von TALK an ein Gerät
Adresse: $FFB4 (65460)

Beschreibung: Entsprechend der Routine LISTN sendet diese Routine das Kommando TALK an ein beliebiges Gerät. Die Geräteadresse ist im <Akku> zu übergeben. Das Kommando

TALK fordert ein am IEC-Bus angeschlossenes Gerät zum Reden, also zum Senden von Informationen auf.

Eingabeparameter: .A

READST

Zweck: Holen des I/O-Statusbytes
Adresse: $FFB7 (65463)

Beschreibung: Es wird der aktuelle Systemstatus im <Akku> zurückgegeben. Ist die RS232 aktiv, so wird das Statusbyte übergeben und direkt im Speicher gelöscht. Sollten Sie also das Statusbyte öfters benötigen, so speichern Sie es zwischen. Ist ein anderer als der RS232-Kanal geöffnet, so wird das Statusbyte von Adresse $90 übergeben.

Ausgabeparameter: .A

SETLFS

Zweck: Fileparameter setzen
Adresse: $FFBA (65466)

Beschreibung: Diese Routine wird überall dort benötigt, wo man Files öffnen muß. Man übergibt die logische File-nummer im <Akku>, die Geräteadresse im X-Register und die Sekundäradresse im Y-Register. Die Routine speichert diese Werte an den Zeropage-Adressen $B8 bis $BA ab.

Eingabeparameter: .A, .X, .Y

SETNAM

Zweck: Setzen der Filenamenparameter
Adresse: $FFBD (65469)

Beschreibung: In der Routine werden die Informationen für den Filenamen in der Zeropage gespeichert. Diese Angaben sind alle

vor dem Öffnen eines Kanales zu machen. Im <Akku> wird die Länge des Filenamens übergeben, im X-Register das Lo-Byte der Adresse und im Y-Register das Hi-Byte der Adresse, an der der Filename gespeichert ist. Ferner müssen Sie mit der SETBNK-Routine die Konfigurationsindizes für den Filenamen und den zu bearbeitenden Speicherbereich übergeben.

Eingabeparameter: .A, .X, .Y

Beispiel:

```
;Eröffnen eines des Directory-Files auf Diskette
LDA #$0C ;Bereich im RAM-Bank 0
TAX      ;Filename auch in RAM-Bank 0
JSR $FF68 ;SETBNK aufrufen
LDA #$01 ;Logische Filenummer
LDX #$08 ;Geräteadresse
LDY #$00 ;Sekundäradresse für Lesen
JSR $FFBA ;SETLFS
LDA #$01 ;Länge des Filenamens
LDX #$00 ;Lo-Byte der Adresse, an der der
LDY #$10 ;Filename gespeichert ist ($1000)
JSR $FFBD ;SETNAM
JSR $FFC0 ;OPEN - Öffnen des Kanals
```

und an Adresse $1000:

```
01000 24 ....
```

OPEN

Zweck: Öffnen einer Datei
Adresse: $FFC0 (65472)

Beschreibung: Es wird die durch die Routinen SETNAM, SETLFS und SETBNK definierte Datei in die Liste der logischen Filenummern aufgenommen. Erst ab diesem Augenblick können die logischen Filenummern bei den Routinen CKOUT und CHKIN angegeben werden. Beachten Sie, daß Sie maximal neun Files auf einmal öffnen können.

CLOSE

Zweck: Schließen einer logischen Datei
Adresse: $FFC3 (65475)

Beschreibung: Es wird die im <Akku> übergebene logische Datei geschlossen. Dabei werden alle gespeicherten Werte wie Geräteadresse, Sekundäradresse etc. in der dafür vorgesehenen Tabelle gelöscht. Ist die Aktion nicht problemlos verlaufen, so wird das CARRY-Flag gesetzt.

Eingabeparameter: .A
Ausgabeparameter: CARRY

Beispiel:

```
;Beispiel für CLOSE
LDA #$01  ;Schließen der Beispieldatei von SETNAM
JSR $FFC3 ;CLOSE ausführen
BCS Error ;Fehler aufgetreten
```

CHKIN

Zweck: Logische Datei als Eingabekanal definieren
Adresse: $FFC6 (65478)

Beschreibung: Im X-Register wird die logische Dateinummer übergeben, die als Eingabekanal benutzt werden soll. Die angegebene logische Dateinummer muß natürlich bereits mit dem OPEN-Kommando geöffnet worden sein. Wird nach dem Aufruf des CHKIN-Kommandos die BASIN-Routine aufgerufen, so erfolgt die Eingabe nicht von Tastatur, sondern von dem geöffneten Gerät; dies kann beispielsweise die Floppy sein. Zu beachten ist, daß zum Einlesen von Tastatur kein CHKIN notwendig ist, da die Tastatur Standard-Eingabegerät ist. Nach einem CLOSE oder CLRCH ist die Tastatur automatisch wieder das Eingabegerät. Auch bei dieser Routine wird das CARRY als OK-Flag benutzt.

Eingabeparameter: .X
Ausgabeparameter: CARRY

Beispiel:

```
;Einlesen der Directory
JSR DIROP ;OPEN 1,8,0,"$" (selbstdefinierte Routine)
LDX #$01  ;LFN der eröffneten Datei
JSR $FFC6 ;CHKIN ausführen
JSR $FFCF ;BASIN - Zeichen holen
```

CKOUT

Zweck: Logische Datei als Ausgabedatei definieren
Adresse: $FFC9 (65481)

Beschreibung: Entsprechend zu CHKIN definiert diese Routine ein im X-Register zu übergebene Datei als Ausgabedatei. Die Datei muß ordnungsgemäß geöffnet worden sein, beispielsweise würde eine Datei, die mit OPEN 1,8,0,"$" geöffnet wurde und mit CKOUT als Ausgabedatei definiert werden soll, einen Fehler hervorrufen, weil diese Datei zum Lesen und nicht zum Schreiben geöffnet wurde. Nach Definition einer Ausgabedatei ist nicht mehr der Bildschirm, sondern die definierte Datei Ausgabegerät. Alle über BSOUT auzugebenen Zeichen werden an dieses Gerät gesandt. Das CARRY-Flag dient als Fehlermelder. Ist es gelöscht, war die Aktion erfolgreich.

Eingabeparameter: .X
Ausgabeparameter: CARRY

CLRCH

Zweck: Ein/Ausgabekanäle schließen
Adresse: $FFCC (65484)

Beschreibung: Diese Routine löscht evtl. mit CHKIN und/oder CKOUT definierte Ein- und Ausgabedateien. Es wird an das Eingabegerät ein UNTALK und an das Ausgabegerät ein UNLISTEN gesendet. Der Bildschirm ist wieder Ausgabe- und

die Tastatur Eingabegerät. Die Dateien werden nicht geschlossen, es erfolgt also kein CLOSE. Es werden weder Ein- noch Ausgabeparameter übergeben.

BASIN

Zweck: Ein Zeichen von Eingabekanal holen
Adresse: $FFCF (65487)

Beschreibung: Die eröffnete und mit CHKIN als Eingabedatei definierte Datei (sonst Tastatur) übergibt ein Zeichen im <Akku>.

Ausgabeparameter: .A

BSOUT

Zweck: Ein Zeichen auf Ausgabekanal ausgeben
Adresse: $FFD2 (65490)

Beschreibung: Es wird das im <Akku> übergebene Zeichen auf die eröffnete und mit CKOUT als Ausgabedatei definierte Datei ausgegeben. Ist der Bildschirm Ausgabedatei (Default), so wird das ASCII-Zeichen in den darzustellenden POKE-Code umgerechnet (ein recht aufwendiges Verfahren. Interessierte sollten sich den entsprechenden Teil im Kernal im C-Bereich ansehen).

Eingabeparameter: .A

Beispiel:

```
;Wechseln des 40/80-Zeichen-Modus
LDA #$1B  ;<ESC>
JSR BSOUT ;$FFD2, Zeichen ausgeben
LDA #"X"  ;<ESC>-X zum Wechseln des Bildschirmstatus
JSR BSOUT ;ausgeben
```

(Es gibt allerdings eine spezielle Routine, die man anspringen kann)

LOADSP

Zweck: Laden einer Datei in den Speicher
Adresse: $FFD5 (65493)

Beschreibung: Bevor mit LOADSP eine Datei geladen werden kann, muß das Gerät, die Sekundäradresse, der Filename etc. durch die Routinen SETLFS, SETNAM und SETBNK definiert worden sein. Im X- (Lo) und Y-Register (Hi) wird die Adresse angegeben, ab der die zu ladende Datei abgelegt werden soll.

Eingabeparameter: .X, .Y

Beispiel:

```
;Laden eines Overlay o.ä.
JSR PREP  ;SETLFS, SETBNK, SETNAM etc.
LDX #$00  ;Lo-Byte von $1000
LDY #$10  ;Hi-Byte von $1000 (Ladeadresse)
JSR $FFD5 ;Lade Datei ab $1000
```

SAVESP

Zweck: Abspeichern eines Bereiches auf Datei
Adresse: $FFD8 (65496)

Beschreibung: Diese Routine speichert einen Speicherbereich auf eine Datei (Diskette, Kassette) ab. Dazu muß man, wie bei der LOADSP-Routine, zunächst Geräteadresse, Sekundäradresse, RAM-Bank, Filename etc. durch die Routinen SETBNK, SETLFS und SETNAM definieren. Im Akku wird die Zeropage-

Adresse angegeben, an der die Anfangsadresse des abzuspeichernden Bereiches steht. Im X- (Lo) und Y-Register (Hi) wird entsprechend die Endadresse des abzuspeichernden Bereiches angegeben.

Eingabeparameter: .A, .X, .Y, Zeropage

Beispiel:

```
;Abspeichern des Bereiches $1000 bis $1100
JSR PREP  ;SETLFS, SETNAM, SETBNK etc. aufrufen
LDA #$00  ;Lo-Byte von $1000
STA $FC   ;in Zeropage speichern
LDA #$10  ;Hi-Byte von $1000
STA $FD   ;in Zeropage speichern
LDA #$FC  ;der Pointer befindet sich an $FC
LDX #$00  ;Lo-Byte der Endadresse $1100
LDY #$11  ;Hi-Byte der Endadresse $1100
JSR $FFD8 ;SAVESP - Speichern des Bereiches $1000-$1100
```

SETTIM

Zweck: Setzen der Systemuhr TI
Adresse: $FFDB (65499)

Beschreibung: Die Routine setzt die Systemuhr TI, die ab Adresse $A0 definiert ist. Diese Uhr wird von der Kernal-IRQ-Routine gesteuert und ist nicht sehr genau. Legen Sie auf eine genauere Uhr Wert, so benutzen Sie die Timer in den beiden CIAs. (Siehe auch entsprechendes Kapitel) Das höchstwertige Byte der 24-Stunden-Uhr wird im Y-Register übergeben.

Eingabeparameter: .A, .X, .Y

Beispiel:

```
;Rücksetzen der Systemuhr
LDA #$00  ;Rücksetzen bedeutet
TAY       ;auf 0,0,0 setzen
TAX       ;Alle drei Register auf null
JSR $FFDB ;SETTIM
```

RDTIM

Zweck: Auslesen der Systemuhr
Adresse: $FFDE (65502)

Beschreibung: Diese Routine liest die 24-Stunden-Uhr aus und übergibt die drei Bytes den Registern Y (höchstwertig), X und <Akku> (niederwertig).

Ausgabeparameter: .A, .X, .Y

Beispiel:

```
;Auslesen der 24-Stunden-Uhr
JSR $FFDE ;RDTIM aufrufen
STY $FC   ;MSB merken
STX $FD   ;mittleres Byte merken
STA $FE   ;LSB merken
```

STOP

Zweck: Abfrage der Stop-Taste
Adresse: $FFE1 (65505)

Beschreibung: Wenn bis zum letzten IRQ-Aufruf die Stop-Taste betätigt worden ist, so wird das ZERO-Flag gesetzt und es wird ein CLRCH ausgeführt. Wurde die Stop-Taste nicht betätigt, so wird das ZERO-Flag gelöscht.

Ausgabeparameter: ZERO-Flag

Beispiel:

```
;Auf STOP prüfen
JSR $FFE1 ;STOP-Taste gedrückt?
BEQ Jawoll;Ist gedrückt
```

GETIN

Zweck: *Holt ein Zeichen aus Tastaturbuffer oder RS232*
Adresse: $FFE4 (65508)

Beschreibung: Holt von der definierten Eingabedatei ein Zeichen. Ist kein Zeichen bereit gestellt, so wird der <Akku> mit null übergeben.

Ausgabeparameter: .A

CLALL

Zweck: Alle offenen Dateien schließen
Adresse: $FFE7 (65511)

Beschreibung: Alle mittels OPEN eröffneten Dateien werden geschlossen oder besser gelöscht - es wird nämlich kein CLOSE ausgeführt. Beispielsweise bei offenen Floppy-Dateien kann dies sehr ärgerlich sein (WRITE FILE OPEN ERROR ist eine Konsequenz). Ferner wird nach dem Löschen der logischen Dateien ein CLRCH (s.o.) ausgeführt. CLALL ist also mit Vorsicht anzuwenden.

UDTIM

Zweck: Systemuhr anpassen (updaten)
Adresse: $FFEA (65514)

Beschreibung: Diese Routine wird vornehmlich von der IRQ-Routine aufgerufen. Es wird die Drei-Byte-24-Stunden-Uhr um eine Einheit hochgezählt.

SCRORG

Zweck: Größe des aktuellen Fensters holen
Adresse: $FFED (65117)

Beschreibung: Die Routine SCRORG holt die aktuellen Fensterwerte in die Register. Der <Akku> enthält nach dem Aufruf die maximale Spaltenzahl, im Y-Register befindet sich die Anzahl der Zeilen im Fenster und im X-Register die Anzahl der Spalten des Fensters.

Ausgabeparameter: .A, .X, .Y

PLOT

Zweck: Cursor-Position holen/setzen
Adresse: $FFF0 (65120)

Beschreibung: Je nach Zustand des CARRY-Flags wird entweder die Cursorposition geholt oder gesetzt. X- und Y-Register sind auf jeden Fall die Kommunikationsregister. Das Y-Register definiert die Zeile (Erste Zeile im Fenster ist null) und das X-Register die Spalte des Cursors. Ist das CARRY-Flag gesetzt, so wird die aktuelle Cursorpostion im Fenster in X- und Y-Register zurückgegeben.

Eingabeparameter: .X, .Y, CARRY

Beispiel:

```
;Einen Stern (*) in die Fenstermitte setzen
JSR $FFED ;SCRORG aufrufen
TXA       ;Spaltenzahl nach <Akku>
LSR A     ;Division durch zwei (Mitte)
TAX       ;und als Spalte wieder nach X
TYA       ;Zeilenzahl nach <Akku>
LSR A     ;Division durch zwei (Mitte)
TAY       ;und wieder als Zeile nach Y
CLC       ;Gelöschtes Carry=Setzen Cursorposition
JSR $FFF0 ;Setze Cursorposition
LDA #"*"  ;<Akku> mit Stern laden
JSR $FFD2 ;und ausgeben.
```

IOBASE

Zweck: Holt die Basisadresse des I/O-Bereiches
Adresse: $FFF3 (65123)

Beschreibung: Es wird die Adresse des Ein- und Ausgabebereiches in X- (Lo) und Y-Register (Hi) übergeben. Diese Adresse ist beim C128 natürlich immer $D000. Für spätere Erweiterungen bzw. Verschiebungen ist es aus Kompatibilitäts-

gründen ratsam, diese Routine in die Software mit zu integrieren und sich darauf zu beziehen.

Ausgabeparameter: .X, .Y

Beispiel:

```
;Anfang des Programmes:
JSR $FFF3 ;IOBASE
STX $FD   ;Lo-Byte merken
STY $FE   ;Hi-Byte merken
```

Im Programm bezieht man diese Adresse dann wie folgt ein:

```
STA ($FD),Y ;In I/O-Bereich
```

7.4.2 Andere nützliche Kernal-Routinen

Es gibt noch einige weitere Routinen im Kernal, die Zeit und Programmspeicher sparen helfen. Vornehmlich befinden sich diese Routinen im Bereich $C000 bis $CFFF und dienen der Ein-/Ausgabe auf den beiden Bildschirmen. Wir stellen Ihnen nun noch einige - unserer Meinung recht nützliche - Routinen in ähnlicher Form wie in Kapitel 7.4.1 vor.

CLRWIN

Zweck: Löschen des Fensters (Bildschirms)
Adresse: $C142 (49474)

Beschreibung: Ist kein Fenster definiert, so wird der gesamte Bildschirm gelöscht. Sollte ein Fenster definiert sein, so wird lediglich innerhalb der Grenzen das definierte Fenster gelöscht.

CURHOM

Zweck: Cursor in HOME-Position im Fenster
Adresse: $C150 (49482)

Beschreibung: Der Cursor wird in die linke obere Ecke des Fensters positioniert. Ist kein Fenster definiert, so kommt der Cursor in die linke obere Ecke des Bildschirmes. Beachten Sie, daß die Position 0/0 immer die linke Ecke des Fensters definiert!

GETLIN

Zweck: Holen einer Eingabezeile
Adresse: $C258 (49752)

Beschreibung: Von Tastatur werden solange Zeichen geholt und auf dem Bildschirm an der aktuellen Cursorpostion dargestellt, bis die Taste <RETURN> gedrückt wird.

BSOUT SCRN

Zweck: Ausgabe eines Zeichens auf akt. Bildschirm
Adresse: $C72D (50989)

Beschreibung: Diese Routine ist die Fortsetzung der BSOUT-Routine an $FFD2, allerdings spart sie sich einige Abfragen, bis man zur Bildschirmausgabe gelangt (ist also somit schneller). Das Zeichen wird im <Akku> übergeben und auf dem momentan aktivem Bildschirm ausgegeben - an der aktuellen Cursor-Position.

Eingabeparameter: .A

CLQIR

Zweck: Löschen von Quote-, Insert- und Revers-Mode
Adresse: $C77D (51069)

Beschreibung: Es werden die Flags für Anführungszeichen-, Insert- und Revers-Modus gelöscht. Die Routine arbeitet estwas schneller, als wenn man erst die dazu notwendigen Steuersequenzen über BSOUT ausgibt.

Es folgen nun noch einige interessante Routinen, die wir lediglich mit Adresse und Wirkung aufführen wollen.

$C854	(51284)	Cursor rechts im Fenster
$C85A	(51290)	Cursor runter im Fenster
$C867	(51303)	Cursor hoch im Fenster
$C875	(51317)	Cursor links im Fenster
$C880	(51328)	Zweiten zeichensatz einschalten
$C8BF	(51391)	RVS-Modus löschen
$C8C1	(51393)	RVS-Modus setzen
$C8C7	(51399)	Unterstreichen einschalten
$C8CE	(51406)	Unterstreichen ausschalten
$C91B	(51483)	Zeichen links von Cursor löschen
$C93D	(51517)	Zeichen unter Cursor löschen
$C94F	(51535)	Tabulator anspringen
$C980	(51584)	Alle Tabulatoren löschen
$C98E	(51598)	BELL - Ton erklingen lassen
$CA14	(51732)	Cursorpos. definiert links/oben von Fenster
$CA16	(51734)	Cursorpos. definiert rechts/oben von Fenster
$CA24	(51748)	Bildschirm als Fenster definieren
$CA52	(51794)	Laufende Zeile löschen
$CA76	(51830)	Cursor bis Zeilenende löschen
$CA8B	(51851)	Zeilenanfang bis Cursorpos. löschen
$CA9F	(51871)	Lösche Cursorpos. bis Bildschirmende
$CABC	(51900)	Aufwärts scrollen
$CAF2	(51954)	Blockcursor einschalten
$CAFE	(51966)	Underline-Cursor einschalten
$CB0B	(51979)	Cursor blinken aus
$CB21	(52001)	Cursor blinken ein
$CB3F	(52031)	80-Zeichen-Monitor negativ darstellen
$CB48	(52040)	80-Zeichen-Monitor normal darstellen
$CC27	(52263)	<Space> an aktuelle Cursorpos.
$CC2F	(52271)	Zeichen <Akku> an akt. Cursorpos.
$CC4A	(52298)	Zeichen <Akku>, <X>:Farbe, <Y>:Spalte auf 80-Zeichen-Bildschirm ausgeben (ohne Cursorbewegung)
$CC6A	(52330)	Hole/Setze Cursorpos.
$CD2C	(52524)	SWAPPER - Umschalten 40/80-Zeichen

8. Das Kernal-ROM

8.1 Einführung

Nachdem Sie sich in den vorangehenden Kapiteln über die grundlegenden Dinge des C128 informiert haben, folgt hier nun das ROM-Listing des Kernals. Wir haben das ROM-Listing bewußt unterteilt in *Kernal-Listing* und *BASIC-ROM*. Wir sind der Meinung, daß dies der Übersichtlichkeit dient. Ferner finden Sie im Kapitel 9 alles neben dem BASIC-ROM-Listing wichtige für BASIC.

Für die eingefleischten Maschinensprache-Programmierer unter Ihnen ist das ROM-Listing wohl eines der wichtigsten Hilfsmittel überhaupt. Für diejenigen unter Ihnen, denen *ROM-Listing* noch kein Begriff ist: Disassembliert man das ROM des Rechners und teilt man Tabellen und Programmcode voneinander, so hat man erst einmal das *nackte* ROM-Listing. Dann beginnt die Arbeit des Dokumentierens, die das Lesen des ROM-Listings später erheblich vereinfacht.

Unter *Kernal* verstehen Sie bitte die wichtigsten elementaren Routine wie Bildschirmein- und -ausgabe, Maschinensprachemonitor, Steuerung des Kassettenlaufwerks etc.

Wir haben uns Mühe gegeben, das ROM-Listing so vollständig und genau wie möglich zu dokumentieren. Hier das Ergebnis:

```
B000:  4C 21 B0    JMP  $B021       Regulärer Monitor Einsprung
B003:  4C 09 B0    JMP  $B009       Monitor  BREAK  Einsprung
B006:  4C B2 B0    JMP  $B0B2       Exmon Monitor Einsprung
B009:  20 7D FF    JSR  $FF7D       Kernal PRIMM: Zeichenkette ausgeben

******************************      Einschaltmeldung des Monitors bei
                                    durch BREAK erzeugtem Einsprung

B00C:  0D 42 52 45 41 4B 07 00      <Cr>   BREAK   <Bell>

******************************      Monitor Initialisierung nach BREAK
                                    Einsprung

B014:  68          PLA              Die auf dem Stack abgelegte BANK Nr.
B015:  85 02       STA  * $02       in d. entspr. Zero Page Byte bringen
B017:  A2 05       LDX  # $05       Hole nacheinander die auf dem Stack
B019:  68          PLA              gesicherten Inhalte des X-Reg,
B01A:  95 03       STA  * $03,X     Y-Reg, Akku, Prozessor Status und
B01C:  CA          DEX              den Programm-Counter Inhalt und
B01D:  10 FA       BPL  $B019       bringe sie in die entsp. ZP Bytes
B01F:  30 25       BMI  $B046       Sprung z. generellen Initialisierung

******************************      Initialisierung bei regulärem Einsp.

B021:  A9 00       LDA  # $00       Configuration Register mit $00 laden
B023:  8D 00 FF    STA  $FF00       und alle System ROMs einschalten
B026:  85 06       STA  * $06       Zero-Page Speicher für Akku löschen
B028:  85 07       STA  * $07       Zero-Page Speicher für X-Reg löschen
B02A:  85 08       STA  * $08       Zero-Page Speicher für Y-Reg löschen
B02C:  85 05       STA  * $05       Speicher f. Prozessor Status löschen
B02E:  A9 00       LDA  # $00       Akku mit Lo-Adr für Monitor laden
B030:  A0 B0       LDY  # $B0       Y-Reg mit Hi-Adr für Monitor laden
B032:  85 04       STA  * $04       Akku in Speicher: Program Counter Lo
B034:  84 03       STY  * $03       Y-Reg in Speicher:Program Counter Hi
B036:  A9 0F       LDA  # $0F       Zero Page Speicher für BANK Nr. auf
B038:  85 02       STA  * $02       $0F= Kernal+Basic, Ram 0, I/O setzen
B03A:  20 7D FF    JSR  $FF7D       Kernal PRIMM: Zeichenkette ausgeben

******************************      Textkonstante für die Monitor
                                    Einschaltmeldung

B03D:  0D 4D 4F 4E 49 54 4F 52      <Cr>   MONITOR
B045:  00

******************************      Generelle Monitor Initialisierung

B046:  D8          CLD              Dezimale Betriebsart zurücksetzen
B047:  BA          TSX              Stack Pointer Inhalt in X-Reg und in
```

```
B048:  86 09      STX  * $09     Speicher für Stack Pointer sichern
B04A:  A9 C0      LDA  # $C0     System und Steuermeldungen zulassen
B04C:  20 90 FF   JSR  $FF90     Kernal SETMSG: Syst/Steuer-Meldungen
B04F:  58         CLI            Alle System Interrupts freigeben

******************************   Monitor Befehl: R (Registerinhalte)

B050:  20 7D FF   JSR  $FF7D     Kernal PRIMM: Zeichenkette ausgeben

******************************   Textkonstante für Prozessorspeicher

B053:  0D 20 20 20 20 50 43 20   <Cr>  PC  SR AC XR YR SP  <Cr>
B05B:  20 53 52 20 41 43 20 58
B063:  52 20 59 52 20 53 50 0D
B06B:  3B 20 1B 51 00            ;  <Esc - Q>

******************************   Gibt den Inhalt der Register u.d.
                                 Stack u. Program-Counter Status aus

B070:  A5 02      LDA  * $02     Hole aktuelle BANK Nr. in Akku
B072:  20 D2 B8   JSR  $B8D2     Akku in 2 Byte ASCII Code: Hi=A,Lo=X
B075:  8A         TXA            ASCII Code f. untere Tetrade in Akku
B076:  20 D2 FF   JSR  $FFD2     Kernal BSOUT: Ein Zeichen ausgeben
B079:  A5 03      LDA  * $03     Zero Page Speicher f. PC-Hi in Akku
B07B:  20 C2 B8   JSR  $B8C2     Akku in 2 Byte ASCII und ausgeben
B07E:  A0 02      LDY  # $02     Displ. zeigt auf ZP Byte f. PC Lo
B080:  B9 02 00   LDA  $0002,Y   PC Lo, P, A, X, Y, S in Akku holen
B083:  20 A5 B8   JSR  $B8A5     Akku als 2 Byte ASCII +<Blank> ausg.
B086:  C8         INY            Displ. erhöhen
B087:  C0 08      CPY  # $08     Bytes $04 - $09 schon ausgegeben?
B089:  90 F5      BCC  $B080     nein, dann nächstes Byte lesen
B08B:  20 B4 B8   JSR  $B8B4     Zeilenvorschub + Clear Rest of Line
B08E:  A2 00      LDX  # $00     Displacement Zeiger für Eingabe-
B090:  86 7A      STX  * $7A     Buffer auf 0 zurücksetzen
B092:  20 CF FF   JSR  $FFCF     Kernal BASIN: Ein Zeichen einlesen
B095:  9D 00 02   STA  $0200,X   und in Monitor E-Buffer ablegen
B098:  E8         INX            Displ. auf E-Buffer erhöhen
B099:  E0 A1      CPX  # $A1     Wurden schon 160 Zeichen eingegeben?
B09B:  B0 1F      BCS  $B0BC     Ja, dann <?> Fehlermeldung ausgeben
B09D:  C9 0D      CMP  # $0D     Wurde <RETURN> eingegeben?
B09F:  D0 F1      BNE  $B092     Nein, dann warte a. nächstes Zeichen
B0A1:  A9 00      LDA  # $00     Wenn <RETURN> eingegeben wurde, dann
B0A3:  9D FF 01   STA  $01FF,X   markiere Befehls-String Ende mit $00
B0A6:  20 E9 B8   JSR  $B8E9     Teste E-Buffer auf Bef-End, <:>, <?>
B0A9:  F0 E0      BEQ  $B08B     War es <:>,<?>,Bef-End zur E-Wartes.
B0AB:  C9 20      CMP  # $20     War Zeichen ein <Blank>?
B0AD:  F0 F7      BEQ  $B0A6     Weiter und nächstes Zeichen lesen
B0AF:  6C 2E 03   JMP  ($032E)   Vektor zeigt auf MONITOR Routine
```

```
B0B2:  A2 15       LDX  # $15       Anzahl der Schlüsselworte in X-Reg
B0B4:  DD E6 B0    CMP  $B0E6,X     mit Schlüsselworttab. abwärts vergl.
B0B7:  F0 0C       BEQ  $B0C5       Wenn gefunden, gehe zur Auswertung
B0B9:  CA          DEX              Zeiger auf Schlüsselworttab. - 1
B0BA:  10 F8       BPL  $B0B4       Schleifen, bis Tab. durchsucht ist
B0BC:  20 7D FF    JSR  $FF7D       Kernal PRIMM: Zeichenkette ausgeben

******************************      ? Konstante f. Monitor Fehlermeldung

B0BF:  1D 3F 00                     <Crsr Right> ?

******************************      Rücksprung in Eingabe Warteschleife

B0C2:  4C 8B B0    JMP  $B08B       Springe in Eingabe Warteschleife

******************************      Adresse d. Monitor Befehls ermitteln

B0C5:  E0 13       CPX  # $13       Ist Schlüsselwort ein <L>, <S>, <V>?
B0C7:  B0 12       BCS  $B0DB       Ja, dann in die entsprechende Ausw.
B0C9:  E0 0F       CPX  # $0F       Ist Schlüsselwort ein Umrechnungsz.?
B0CB:  B0 13       BCS  $B0E0       ($,+,&,%) Ja, dann in entspr. Ausw.
B0CD:  8A          TXA              Schlüsselwort-Nummer in Akku bringen
B0CE:  0A          ASL  A           und mit 2 multiplizieren
B0CF:  AA          TAX              diesen Wert als Offset in X-Reg
B0D0:  BD FD B0    LDA  $B0FD,X     Adr. der Monitor Routine (Hi) holen
B0D3:  48          PHA              und als quasi RTS auf Stack ablegen
B0D4:  BD FC B0    LDA  $B0FC,X     Adr. der Monitor Routine (Lo) holen
B0D7:  48          PHA              und als quasi RTS auf Stack ablegen
B0D8:  4C A7 B7    JMP  $B7A7       Zur Befehlsparameter Auswertung

******************************      Auskoppeln von LSV und Umrechnungen

B0DB:  85 93       STA  * $93       Zeichen f. Bef-Schlüsselwort sichern
B0DD:  4C 37 B3    JMP  $B337       Zur Auswertung von LSV-Befehlen
B0E0:  4C B1 B9    JMP  $B9B1       Zur Auswertung v. Umrechnungszeichen

******************************      Monitor Befehl: X  (Exit)

B0E3:  6C 00 0A    JMP  ($0A00)     Vektor: Basic Warm-Start  ($4003)

******************************      Monitor - Schlüsselworte

B0E6:  41 43 44 46 47 48 4A 4D      A C D F G H J M
B0EE:  52 54 58 40 2E 3E 3B 24      R T X @ . > ; $
B0F6:  2B 26 25 4C 53 56            + & % L S V

******************************      Adressen der Monitor Befehle (-1)
```

```
B0FC:  05 B4      ($B406)        A = Assemble
B0FE:  30 B2      ($B231)        C = Compare
B100:  98 B5      ($B599)        D = Disassemble
B102:  DA B3      ($B3DB)        F = Fill
B104:  D5 B1      ($B1D6)        G = Go to
B106:  CD B2      ($B2CE)        H = Hunt
B108:  DE B1      ($B1DF)        J = Jump
B10A:  51 B1      ($B152)        M = Monitor
B10C:  4F B0      ($B050)        R = Register
B10E:  33 B2      ($B234)        T = Transfer
B110:  E2 B0      ($B0E3)        X = Exit
B112:  8F BA      ($BA90)        @ = Disc Command
B114:  05 B4      ($B406)        . = Assemble
B116:  AA B1      ($B1AB)        > = Modify Memory
B118:  93 B1      ($B194)        ; = Modify Register

******************************   LDA Routine für Akku aus bel. Bank
                                 FETVEC = Bank Byte des OP3 Operanden

B11A:  8E B2 0A   STX  $0AB2     X-Reg zwischenspeichern
B11D:  A6 68      LDX  * $68     Bank Nr. aus 'Von' Operand OP3 holen
B11F:  A9 66      LDA  # $66     FETVEC Adresse f. Indfet Rout. in A
B121:  78         SEI            Alle System Interrupts verhindern
B122:  20 74 FF   JSR  $FF74     Kernal INDFET:LDA(fetvec),Y bel.Bank
B125:  58         CLI            Alle System Interrupts freigeben
B126:  AE B2 0A   LDX  $0AB2     X-Reg mit gesichertem Wert laden
B129:  60         RTS            Rücksprung aus dem Unterprogramm

******************************   STA Routine für Akku Inhalt in bel.
                                 Bank. STAVEC = Bank Byte des OP3

B12A:  8E B2 0A   STX  $0AB2     X-Reg zwischenspeichern
B12D:  A2 66      LDX  # $66     STAVEC Lo Adresse in X-Reg laden und
B12F:  8E B9 02   STX  $02B9     in STAVEC Speicher f. Indsta Routine
B132:  A6 68      LDX  * $68     Bank Nr. aus 'Von'Operand OP3 holen
B134:  78         SEI            Alle System Interrupts verhindern
B135:  20 77 FF   JSR  $FF77     Kernal INDSTA:STA(stavec),Y bel.Bank
B138:  58         CLI            Alle System Interrupts freigeben
B139:  AE B2 0A   LDX  $0AB2     X-Reg mit gesichertem Wert laden
B13C:  60         RTS            Rücksprung aus dem Unterprogramm

******************************   CMP Routine des Akku-Inhalts mit bel.
                                 Bank. CMPVEC = Bank Byte des OP3

B13D:  8E B2 0A   STX  $0AB2     X-Reg zwischenspeichern
B140:  A2 66      LDX  # $66     CMPVEC Lo Adresse in X-Reg laden und
B142:  8E C8 02   STX  $02C8     in CMPVEC Speicher f. Indcmp Routine
B145:  A6 68      LDX  * $68     Bank Nr. aus 'Von' Operand OP3 holen
```

```
B147:  78        SEI            Alle System Interrupts verhindern
B148:  20 7A FF  JSR  $FF7A     Kernal INDCMP:CMP(cmpvec),Y bel.Bank
B14B:  58        CLI            Alle System Interrupts freigeben
B14C:  08        PHP            Ergebnis des Vergleichs sichern
B14D:  AE B2 0A  LDX  $0AB2     X-Reg mit gesichertem Wert laden
B150:  28        PLP            Ergebnis des Vergleichs zurückholen
B151:  60        RTS            Rücksprung aus dem Unterprogramm

******************************  Monitor Befehl: M  (Memory display)

B152:  B0 08     BCS  $B15C     Kein Parameter, dann Standard setzen
B154:  20 01 B9  JSR  $B901     Kopieren des OP1 Inhalts nach OP3
B157:  20 A7 B7  JSR  $B7A7     Hole 'Bis' Operanden in OP1
B15A:  90 06     BCC  $B162     Bei 'Von'- 'Bis' Stepzahl ermitteln
B15C:  A9 0B     LDA  # $0B     Adr-Lo des OP1 Operanden m. Standard
B15E:  85 60     STA  * $60     Stepzahl 12 laden
B160:  D0 15     BNE  $B177     Z. Ausführung d. Memory Display Bef.
B162:  20 0E B9  JSR  $B90E     Differenz: OP1-OP3 in OP1 speichern
B165:  90 2A     BCC  $B191     'Von' Adr größer 'Bis' Adr = Fehler
B167:  A2 03     LDX  # $03     Stepzahl 3mal durch 2 teilen
B169:  24 D7     BIT  * $D7     Teste auf 40/80 Zeichen Bildschirm
B16B:  10 01     BPL  $B16E     Bei 40 Zeichen zur Step Division
B16D:  E8        INX            Bei 80 Zeichen Bildsch. Stepzahl/16
B16E:  46 62     LSR  * $62     Division des 3 Byte Operanden OP1
B170:  66 61     ROR  # $61     durch Faktor 2, da je Mem.Disp.Zeile
B172:  66 60     ROR  # $60     8 oder 16 Werte angezeigt werden
B174:  CA        DEX            Divisionszähler für Stepzahl -1
B175:  D0 F7     BNE  $B16E     OP1 Operand schon durch 8/16 geteilt
B177:  20 E1 FF  JSR  $FFE1     Kernal STOP: Auf Stop Taste prüfen
B17A:  F0 12     BEQ  $B18E     Wenn Stop gedrückt, dann Exit Rout.
B17C:  20 E8 B1  JSR  $B1E8     Eine Memory Display Zeile ausgeben
B17F:  A9 08     LDA  # $08     Additionskonstante f. 'Von' Operand
B181:  24 D7     BIT  * $D7     Teste auf 40/80 Zeichen Bildschirm
B183:  10 01     BPL  $B186     Bei 40 Zeichen ist Additionsk. 8 ok
B185:  0A        ASL  A         Bei 80 Zeichen Additionsk. * 2 (=16)
B186:  20 52 B9  JSR  $B952     Addition: Akku Inhalt zum OP3
B189:  20 22 B9  JSR  $B922     Subtraktion: OP1 - Konstante <1>
B18C:  B0 E9     BCS  $B177     Schleifen, bis OP1 kleiner 0 ist
B18E:  4C 8B B0  JMP  $B08B     Springe in Eingabe Warteschleife
B191:  4C BC B0  JMP  $B0BC     <?> Ausgeben und zur E-Warteschleife

******************************  Monitor Befehl: ;  (Modify Reg)

B194:  20 74 B9  JSR  $B974     Bei C=0 OP1 in Z-Page Bank/PCHi/PCLo
B197:  A0 00     LDY  # $00     Displacement für Zero-Page setzen
B199:  20 A7 B7  JSR  $B7A7     Hole Modify Wert in OP1 Operand
B19C:  B0 0A     BCS  $B1A8     Carry gesetzt = KZ f. Exit Routine
B19E:  A5 60     LDA  * $60     Hole Adr-Lo von OP1 als Modify Wert
```

```
B1A0:  99 05 00   STA  $0005,Y    Modify Status-B.,A,X,Y,Stack Pointer
B1A3:  C8         INY             Displ. auf Z-Page CPU Speicher + 1
B1A4:  C0 05      CPY  # $05      Alle 4 CPU Speicher schon verändert?
B1A6:  90 F1      BCC  $B199      Nein, dann z. nächsten Reg.-Änderung
B1A8:  4C 8B B0   JMP  $B08B      Springe in Eingabe Warteschleife

******************************    Monitor Befehl: >  (Modify Mem)

B1AB:  B0 1C      BCS  $B1C9      Kein Parameter, dann keine Änderung
B1AD:  20 01 B9   JSR  $B901      Kopieren des OP1 Inhalts nach OP3
B1B0:  A0 00      LDY  # $00      Modify Displ. Zeiger auf 0 setzen
B1B2:  20 A7 B7   JSR  $B7A7      Hole Modify Wert in Operand OP1
B1B5:  B0 12      BCS  $B1C9      Kein weiterer Wert = Zeile ausgeben
B1B7:  A5 60      LDA  * $60      Mod. Wert aus Adr-Lo (OP1) holen
B1B9:  20 2A B1   JSR  $B12A      STA Routine in beliebige Bank
B1BC:  C8         INY             Displ. Zeiger für Modify Byte + 1
B1BD:  24 D7      BIT  * $D7      Teste auf 40/80 Zeichen Bildschirm
B1BF:  10 04      BPL  $B1C5      Zur Parameter Max Abfrage bei 40 Z.
B1C1:  C0 10      CPY  # $10      Schon 16 Zeichen gelesen/geändert?
B1C3:  90 ED      BCC  $B1B2      Nein, hole nächsten Mod. Parameter
B1C5:  C0 08      CPY  # $08      Schon 8  Zeichen gelesen/geändert?
B1C7:  90 E9      BCC  $B1B2      Nein, hole nächsten Mod. Parameter
B1C9:  20 7D FF   JSR  $FF7D      Kernal PRIMM: Zeichenkette ausgeben

******************************    Lösche Insert, RVS und Quote Modus

B1CC:  1B 4F 91 00                <Esc - O>  <Crsr Up>

******************************    Ausgabe e. veränderten Memory Zeile

B1D0:  20 E8 B1   JSR  $B1E8      Gibt: < 8/16 Hexwerte,8/16 ASCII aus
B1D3:  4C 8B B0   JMP  $B08B      Springe in Eingabe Warteschleife

******************************    Monitor Befehl: G  (Go to)

B1D6:  20 74 B9   JSR  $B974      Bei C=0 OP1 in Z-Page Bank/PCHi/PCLo
B1D9:  A6 09      LDX  * $09      Lade X m. Z-Page Byte f. Stackpoin.
B1DB:  9A         TXS             Stack Pointer mit X-Reg modifizieren
B1DC:  4C 71 FF   JMP  $FF71      Kernal JMPFAR: JMP in beliebige Bank

******************************    Monitor Befehl: J  (Jump to)

B1DF:  20 74 B9   JSR  $B974      Bei C=0 OP1 in Z-Page Bank/PCHi/PCLo
B1E2:  20 6E FF   JSR  $FF6E      Kernal JSRFAR: JSR belieb. Bank +RTS
B1E5:  4C 8B B0   JMP  $B08B      Springe in Eingabe Warteschleife
```

```
*******************************     Gibt ein '<', 8/16 Hexwerte und 8/16
                                    ASCII Zeichen für Memory Display aus

B1E8:  20 B4 B8   JSR  $B8B4        Zeilenvorschub + Clear Rest of Line
B1EB:  A9 3E      LDA  # $3E        Akku mit '<' Zeichen laden
B1ED:  20 D2 FF   JSR  $FFD2        Kernal BSOUT: Ein Zeichen ausgeben
B1F0:  20 92 B8   JSR  $B892        OP3 in 5-Byte-ASCII-Zeichen ausgeben
B1F3:  A0 00      LDY  # $00        Schleifenzähler auf 0 setzen
B1F5:  F0 03      BEQ  $B1FA        B. 1.Hexwert Skip-Zwischenraum Blank
B1F7:  20 A8 B8   JSR  $B8A8        <Blank>, <Cr>, <Crsr-Up> ausgeben
B1FA:  20 1A B1   JSR  $B11A        LDA Routine für Akku aus bel. Bank
B1FD:  20 C2 B8   JSR  $B8C2        A als 2 Byte ASCII Zeichen ausgeben
B200:  C8         INY               Schleifen + Displacement Zähler + 1
B201:  C0 08      CPY  # $08        Schon 8 Hexwerte ausgegeben?
B203:  24 D7      BIT  * $D7        Teste auf 40/80-Zeichen-Bildschirm
B205:  10 02      BPL  $B209        Bei 40 Zeichen Ausgabe fortfahren
B207:  C0 10      CPY  # $10        Schon 16 Hexwerte ausgegeben?
B209:  90 EC      BCC  $B1F7        Nächsten Hexwert holen
B20B:  20 7D FF   JSR  $FF7D        Kernal PRIMM: Zeichenkette ausgeben

*******************************     Konstante: Doppelpunkt, RVS ein

B20E:  3A 12 00                     :  <Rvs On>

*******************************     Ausgabe von 8/16 Bytes als ASCII

B211:  A0 00      LDY  # $00        Schleifen und Disp.-Zähler auf 0
B213:  20 1A B1   JSR  $B11A        LDA Routine für Akku aus bel. Bank
B216:  48         PHA               Zeichen auf Stack sichern
B217:  29 7F      AND  # $7F        Bit 7 ausblenden (keine RVS-Zeichen)
B219:  C9 20      CMP  # $20        Prüfe, ob es ein Steuerzeichen ist
B21B:  68         PLA               Zeichen wieder vom Stack holen
B21C:  B0 02      BCS  $B220        Wenn kein Steuercode, normal ausgeb.
B21E:  A9 2E      LDA  # $2E        sonst Akkumulator mit <.> laden
B220:  20 D2 FF   JSR  $FFD2        Kernal BSOUT: Ein Zeichen ausgeben
B223:  C8         INY               Schleifen-und Displacement-Zähler +1
B224:  24 D7      BIT  * $D7        Teste auf 40/80 Zeichen Bildschirm
B226:  10 04      BPL  $B22C        Bei 40 Zeichen Anzeige weitermachen
B228:  C0 10      CPY  # $10        Schon 16 Charakter ausgegeben ? (80)
B22A:  90 E7      BCC  $B213        Nein, nächsten Charakter ausgeben
B22C:  C0 08      CPY  # $08        Schon 8 Charakter ausgegeben ?  (40)
B22E:  90 E3      BCC  $B213        Nein, nächsten Charakter ausgeben
B230:  60         RTS               Rücksprung aus dem Unterprogramm

*******************************     Monitor Befehl: C  (Compare)
```

```
B231:  A9 00     LDA  # $00     Kennzeichen für COMPARE setzen
B233:  2C        .Byte  $2C     Skip nach $B236

******************************  Monitor Befehl: T (Transfer)

B234:  A9 80     LDA  # $80     Kennzeichen für TRANSFORM setzen
B236:  85 93     STA  * $93     und in BefehlsByte-Speicher ablegen
B238:  A9 00     LDA  # $00     Richtungszeiger für C/T Bef. auf $00
B23A:  8D B3 0A  STA  $0AB3     (= vorwärts) setzen ($80= rückwärts)
B23D:  20 83 B9  JSR  $B983     Hole 'Bis' und Stepzahl in OPH,OP2
B240:  B0 05     BCS  $B247     Carry gesetzt= KZ f. Fehler gefunden
B242:  20 A7 B7  JSR  $B7A7     Hole 'Nach'/ 'Mit' Operanden in OP1
B245:  90 03     BCC  $B24A     'Nach'/ 'Mit' Operand ist ok
B247:  4C BC B0  JMP  $B0BC     <?> Ausgeben und zur E-Warteschleife
B24A:  24 93     BIT  * $93     War es Transfer(-) oder Compare(+)
B24C:  10 2C     BPL  $B27A     In die Compare Routine
B24E:  38        SEC            Carry für Subtraktion setzen
B24F:  A5 66     LDA  * $66     Prüfe, ob der Inhalt der beiden
B251:  E5 60     SBC  * $60     Adreßbytes (Adr-Lo), (Adr-Hi) des
B253:  A5 67     LDA  * $67     Operanden OP3 größer ist, als die
B255:  E5 61     SBC  * $61     beiden Adreßbytes des Operanden OP1
B257:  B0 21     BCS  $B27A     'Nach' kleiner 'Von' = Richtung ok
B259:  A5 63     LDA  * $63     Addiere den Inhalt des 3 Byte
B25B:  65 60     ADC  * $60     Operanden OP2 in den Speicherstellen
B25D:  85 60     STA  * $60     $65-$64-$63 zum Inhalt
B25F:  A5 64     LDA  * $64     des 3 Byte Operanden OP1
B261:  65 61     ADC  * $61     in den Speicherstellen $62-$61-$60.
B263:  85 61     STA  * $61     Beachte dabei jeweils den evtl.
B265:  A5 65     LDA  * $65     auftretenden AdditionsÜberlauf
B267:  65 62     ADC  * $62     Speichere das Ergebnis der Addition
B269:  85 62     STA  * $62     im Operanden OP1
B26B:  A2 02     LDX  # $02     Kopiere d. Inhalt des
B26D:  BD B7 0A  LDA  $0AB7,X   3-Byte-Hilfsoperanden
B270:  95 66     STA  * $66,X   in den Speicherstellen
B272:  CA        DEX            $0AB9-$0AB8-$0AB7 in den
B273:  10 F8     BPL  $B26D     Operanden OP3 ($68-$67-$66)
B275:  A9 80     LDA  # $80     Wenn 'Nach' größer 'Von' ist, dann
B277:  8D B3 0A  STA  $0AB3     setze Richtungskz auf rückwärts
B27A:  20 B4 B8  JSR  $B8B4     Zeilenvorschub + Clear Rest of Line
B27D:  A0 00     LDY  # $00     Displacement-Zeiger auf 0 setzen
B27F:  20 E1 FF  JSR  $FFE1     Kernal STOP: Auf Stop Taste prüfen
B282:  F0 47     BEQ  $B2CB     Wenn Stop gedrückt, dann Exit Rout.
B284:  20 1A B1  JSR  $B11A     LDA Routine für Akkus aus bel. Bank
B287:  A2 60     LDX  # $60     $60 ist Adr-Lo des 'Mit' 'Nach' OP1
B289:  8E B9 02  STX  $02B9     STAVEC auf diese Adresse setzen
B28C:  8E C8 02  STX  $02C8     CMPVEC auf diese Adresse setzen
B28F:  A6 62     LDX  * $62     X-Reg mit Bank Byte 'Nach' laden
B291:  78        SEI            Alle System Interrupts verhindern
```

```
B292:  24 93     BIT  * $93     War es ein Transfer oder Compare?
B294:  10 03     BPL  $B299     Bei Compare in entsprechende Routine
B296:  20 77 FF  JSR  $FF77     Kernal INDSTA:STA(stavec),Y bel.Bank
B299:  A6 62     LDX  * $62     X-Reg mit Bank Byte 'Mit' laden
B29B:  20 7A FF  JSR  $FF7A     Kernal INDCMP:CMP(cmpvec),Y bel.Bank
B29E:  58        CLI            Alle System Interrupts freigeben
B29F:  F0 09     BEQ  $B2AA     Bei "gleich" nicht ausgeben, sonst
B2A1:  20 92 B8  JSR  $B892     Inhalt OP3 in 5 Byte ASCII ausgeben
B2A4:  20 A8 B8  JSR  $B8A8     <Blank>, <Cr>, <Crsr-Up> ausgeben
B2A7:  20 A8 B8  JSR  $B8A8     <Blank>, <Cr>, <Crsr-Up> ausgeben
B2AA:  2C B3 0A  BIT  $0AB3     Prüfe die Transfer Richtung
B2AD:  30 0B     BMI  $B2BA     Ermitteln der neuen rückwärts Adr.
B2AF:  E6 60     INC  * $60     Für vorwärts Transfer die
B2B1:  D0 10     BNE  $B2C3     'Nach' Adresse um 1 erhöhen und
B2B3:  E6 61     INC  * $61     dabei Überlauf beachten
B2B5:  D0 0C     BNE  $B2C3     Wenn Überlauf in Adr-Hi, dann Fehler
B2B7:  4C BC B0  JMP  $B0BC     <?> ausgeben und zur E-Warteschleife
B2BA:  20 22 B9  JSR  $B922     Subtraktion: OP1 - Konstante <1>
B2BD:  20 60 B9  JSR  $B960     Subtraktion: OP3 - Konstante <1>
B2C0:  4C C6 B2  JMP  $B2C6     Springe zur Subtraktion OP2 - <1>

******************************  Stepzahl und 'Von' Adresse setzen

B2C3:  20 50 B9  JSR  $B950     Addition: Konstante <1> zum OP3
B2C6:  20 3C B9  JSR  $B93C     Subtraktion: OP2 - Konstante <1>
B2C9:  B0 B4     BCS  $B27F     Schleifen, bis alle Steps erledigt
B2CB:  4C 8B B0  JMP  $B08B     Springe in Eingabe Warteschleife

******************************  Monitor Befehl: H  (Hunt)

B2CE:  20 83 B9  JSR  $B983     Hole 'Bis' und Stepzahl in OP1
B2D1:  B0 61     BCS  $B334     Carry gesetzt= KZ f. Fehler gefunden
B2D3:  A0 00     LDY  # $00     Displ. a. Hunt Zeichen im VGL Buffer
B2D5:  20 E9 B8  JSR  $B8E9     Lese ein Zeichen aus Eingabe-Buffer
B2D8:  C9 27     CMP  # $27     War gelesenes Zeichen ein <'>?
B2DA:  D0 16     BNE  $B2F2     Nein, dann keine String Suche
B2DC:  20 E9 B8  JSR  $B8E9     Lese ein Zeichen aus Eingabe-Buffer
B2DF:  C9 00     CMP  # $00     Wurde Bef-End gefunden?
B2E1:  F0 51     BEQ  $B334     Ja, dann Fehler <?> ausgeben
B2E3:  99 80 0A  STA  $0A80,Y   Zeichen im VGL Buffer ablegen
B2E6:  C8        INY            Displacement auf VGL Buffer +1
B2E7:  20 E9 B8  JSR  $B8E9     Teste E-Buffer auf Bef-End, <:>, <?>
B2EA:  F0 1B     BEQ  $B307     Wenn Bef-End,dann in Hunt Ausführung
B2EC:  C0 20     CPY  # $20     Schon 32 Werte im VGL Buffer?
B2EE:  D0 F3     BNE  $B2E3     Nein, dann nächsten VGL Wert holen
B2F0:  F0 15     BEQ  $B307     Zur Ausführung der Hunt Routine
B2F2:  8C 00 01  STY  $0100     Displ. auf VGL Buffer sichern
B2F5:  20 A5 B7  JSR  $B7A5     Hole VGL Operand in OP1 (wie CHRGOT)
```

```
B2F8:  A5 60     LDA  * $60     Ablegen des aus OP1 ermittelten
B2FA:  99 80 0A  STA  $0A80,Y   1 Byte Wertes in den VGL Buffer
B2FD:  C8        INY            Displacement auf VGL-Buffer +1
B2FE:  20 A7 B7  JSR  $B7A7     Hole weiteren VGL Wert in OP1
B301:  B0 04     BCS  $B307     Keiner vorhanden,dann Hunt ausführen
B303:  C0 20     CPY  # $20     Schon 32 Werte im VGL-Buffer?
B305:  D0 F1     BNE  $B2F8     Nein, dann nächsten VGL Wert holen
B307:  84 93     STY  * $93     Zahl d. Werte d. VGL-Buffers sichern
B309:  20 B4 B8  JSR  $B8B4     Zeilenvorschub + Clear Rest of Line
B30C:  A0 00     LDY  # $00     Displ. auf 1. Zeichen im VGL-Buffer
B30E:  20 1A B1  JSR  $B11A     LDA-Routine für Akku aus bel. Bank
B311:  D9 80 0A  CMP  $0A80,Y   Vergleich mit Zeichen aus VGL-Buffer
B314:  D0 0E     BNE  $B324     Ungleich, dann nächster Step
B316:  C8        INY            Displ. auf nächsten Wert im VGL Buf.
B317:  C4 93     CPY  * $93     Alle Einzelvergleiche durchgeführt ?
B319:  D0 F3     BNE  $B30E     Nein, nächster Vergleich dies. Steps
B31B:  20 92 B8  JSR  $B892     Inhalt OP3 als 5 Byte ASCII ausgeben
B31E:  20 A8 B8  JSR  $B8A8     <Blank>, <Cr>, <Crsr-Up> ausgeben
B321:  20 A8 B8  JSR  $B8A8     <Blank>, <Cr>, <Crsr-Up> ausgeben
B324:  20 E1 FF  JSR  $FFE1     Kernal STOP: Auf Stop Taste prüfen
B327:  F0 08     BEQ  $B331     Wenn Stop gedrückt, dann Exit-Rout.
B329:  20 50 B9  JSR  $B950     Addition: Konstante <1> zum OP3
B32C:  20 3C B9  JSR  $B93C     Subtraktion: OP2 - Konstante <1>
B32F:  B0 DB     BCS  $B30C     Schleifen, bis alle Steps erledigt
B331:  4C 8B B0  JMP  $B08B     Springe in Eingabe-Warteschleife
B334:  4C BC B0  JMP  $B0BC     <?> Ausgeben und zur E-Warteschleife

*******************************  Einsprung für die Monitor Befehle:
                                L = Load, S = Save, V = Verify

B337:  A0 01     LDY  # $01     Y-Reg mit $01 laden
B339:  84 BA     STY  * $BA     Geräteadresse setzen (1=Datassette)
B33B:  84 B9     STY  * $B9     Sekundäradresse setzen (1=schreiben)
B33D:  88        DEY            Y-Reg auf $00 herunterzählen
B33E:  84 C6     STY  * $C6     Bank-Nr. für LSV-Aufrufe setzen
B340:  84 B7     STY  * $B7     Länge des Dateinamens auf 0 setzen
B342:  84 C7     STY  * $C7     Bank-Nr. für Adr. Dateiname setzen
B344:  84 90     STY  * $90     Statusbyte löschen (0 = alles OK)
B346:  A9 0A     LDA  # $0A     Zero Page Speicher für Adr. Hi des
B348:  85 BC     STA  * $BC     aktuellen Filenamens mit $0A laden
B34A:  A9 80     LDA  # $80     Zero Page Speicher für Adr. Lo des
B34C:  85 BB     STA  * $BB     Filenamens mit $80 laden (= $0A80)
B34E:  20 E9 B8  JSR  $B8E9     Teste E-Buffer auf Bef-End, <:>, <?>
B351:  F0 58     BEQ  $B3AB     Wenn <:>,<?>,Bef-End in E-Warteschl.
B353:  C9 20     CMP  # $20     War gelesenes Zeichen ein <Blank>?
B355:  F0 F7     BEQ  $B34E     Ja, dann zurück und weiterlesen
B357:  C9 22     CMP  # $22     War gelesenes Zeichen ein <">?
B359:  D0 15     BNE  $B370     Nein, dann Fehler im Befehlsstring
```

```
B35B:  A6 7A     LDX  * $7A      X-Reg mit Displ. auf E-Buffer laden
B35D:  BD 00 02  LDA  $0200,X    lese E-Buffer ab 1.<"> (= Dateiname)
B360:  F0 49     BEQ  $B3AB      $00 = Ende Befehlsstring
B362:  E8        INX             E-Buffer Zeiger auf nächstes Zeichen
B363:  C9 22     CMP  # $22      Wurde 2. <"> gefunden ?
B365:  F0 0C     BEQ  $B373      Ja, dann weitere Auswertung
B367:  91 BB     STA  ($BB),Y    Dateiname ab $0A80 ablegen
B369:  E6 B7     INC  * $B7      Zähler für Länge des Dateinamens + 1
B36B:  C8        INY             Zeiger auf Filename-Speicher erhöhen
B36C:  C0 11     CPY  # $11      Dateiname länger als 16 Zeichen ?
B36E:  90 ED     BCC  $B35D      Nein, dann nächstes Zeichen lesen
B370:  4C BC B0  JMP  $B0BC      <?> Ausgeben und zur E-Warteschleife

******************************   LSV-Parameter-Auswertung nach 2.<">

B373:  86 7A     STX  * $7A      Displ. Zeiger E-Buf zeigt hinter 2."
B375:  20 E9 B8  JSR  $B8E9      Teste E-Buffer auf Bef-End, <:>, <?>
B378:  F0 31     BEQ  $B3AB      Ohne Parameter kann gültiger LV sein
B37A:  20 A7 B7  JSR  $B7A7      Parameter in OP1 holen (hier GA)
B37D:  B0 2C     BCS  $B3AB      Kein Parameter,dann z. LV-Auswertung
B37F:  A5 60     LDA  * $60      Adr-Lo (OP1) holen (hier Geräteadr.)
B381:  85 BA     STA  * $BA      und in Z-Page Speicher f. GA bringen
B383:  20 A7 B7  JSR  $B7A7      Parameter in OP1 holen (Startadr.)
B386:  B0 23     BCS  $B3AB      Kein Parameter,dann z. LV-Auswertung
B388:  20 01 B9  JSR  $B901      Kopieren des OP1 Inhalts nach OP3
B38B:  85 C6     STA  * $C6      Bank.Nr. (OP1) in Z-Page Bank-B.,LSV
B38D:  20 A7 B7  JSR  $B7A7      Parameter in OP1 holen (Endadresse)
B390:  B0 3F     BCS  $B3D1      Kein Parameter,dann z. LV-Auswertung
B392:  20 B4 B8  JSR  $B8B4      Zeilenvorschub + Clear Rest of Line
B395:  A6 60     LDX  * $60      Adr-Lo (OP1) ist 'Bis' Wert f. SAVE
B397:  A4 61     LDY  * $61      Adr-Hi (OP1) ist 'Bis' Wert f. SAVE
B399:  A5 93     LDA  * $93      Hole gesichertes Bef-Schlüsselwort
B39B:  C9 53     CMP  # $53      War es ein <S> für Save?
B39D:  D0 D1     BNE  $B370      Nein = Fehler, da kein 'Bis' für LV
B39F:  A9 00     LDA  # $00      Akku mit 0 laden und in Zero-Page
B3A1:  85 B9     STA  * $B9      Speicher für Sekundäradresse bringen
B3A3:  A9 66     LDA  # $66      Bank Nr. aus 'Von' Operand (OP3)
B3A5:  20 D8 FF  JSR  $FFD8      Kernal SAVESP: Datei abspeichern
B3A8:  4C 8B B0  JMP  $B08B      Springe in Eingabe Warteschleife

******************************   Ausführung gültiger LV-Befehle

B3AB:  A5 93     LDA  * $93      gesichertes Bef.-Schlüsselwort holen
B3AD:  C9 56     CMP  # $56      War es ein <V> für Verify?
B3AF:  F0 06     BEQ  $B3B7      Akku <> 0 ist Verify Kz. in LOADSP
B3B1:  C9 4C     CMP  # $4C      War es ein <L> für Load?
B3B3: D0BB     BNE  $B370     Nein,dann war es <S>f. Save Befehl
B3B5:  A9 00     LDA  # $00      u = 0 ist Load-Kennz. in LOADSP
```

```
B3B7:  20 D5 FF   JSR  $FFD5     Kernal LOADSP: Datei einladen
B3BA:  A5 90      LDA  * $90     System-STATUS in Akku laden
B3BC:  29 10      AND  # $10     Bit für Lesefehler ausmaskieren
B3BE:  F0 E8      BEQ  $B3A8     Kein Fehler bei LV. Zur E-Warteschl.
B3C0:  A5 93      LDA  * $93     Zeichen f. Bef-Schlüsselwort holen
B3C2:  F0 AC      BEQ  $B370     Kein Bef-Schlüsselwort, dann Fehler
B3C4:  20 7D FF   JSR  $FF7D     Kernal PRIMM: Zeichenkette ausgeben

******************************   Monitor-Konstante für <ERROR>

B3C7:  20 45 52 52 4F 52 00      ERROR

******************************   Nach Error-Ausgabe in E-Warteschl.

B3CE:  4C 8B B0   JMP  $B08B     Springe in Eingabe-Warteschleife

******************************   Vorbereitung für gültige LV-Befehle
                                 mit Geräte- und Startadresse

B3D1:  A6 66      LDX  * $66     Adr-Lo (OP3) f.Startadresse in X-Reg
B3D3:  A4 67      LDY  * $67     Adr-Hi (OP3) f.Startadresse in Y-Reg
B3D5:  A9 00      LDA  # $00     schreibe Sekundäradresse $00 = Lesen
B3D7:  85 B9      STA  * $B9     in Z-Page-Speicher f. Sek-Adr
B3D9:  F0 D0      BEQ  $B3AB     Zur Ausführung gültiger LV Befehle

******************************   Monitor-Befehl: F  (Fill)

B3DB:  20 83 B9   JSR  $B983     Hole 'Bis' und Stepzahl in OPH,OP2
B3DE:  B0 23      BCS  $B403     Carry gesetzt = KZ für Fehlerausgabe
B3E0:  A5 68      LDA  * $68     Hole Bank-Nr. d. 'Von' Operanden OP3
B3E2:  CD B9 0A   CMP  $0AB9     Vergl. mit Bank-Nr. des 'Bis' Oper.
B3E5:  D0 1C      BNE  $B403     Ungleich ist  = KZ für Fehlerausgabe
B3E7:  20 A7 B7   JSR  $B7A7     Hole Befehlsparameter (Fill-Wert)
B3EA:  B0 17      BCS  $B403     Carry gesetzt = KZ für Fehlerausgabe
B3EC:  A0 00      LDY  # $00     Displ. für Fill-Befehl auf 0 setzen
B3EE:  A5 60      LDA  * $60     in Adr-Lo (OP1) steht der Fill-Wert
B3F0:  20 2A B1   JSR  $B12A     STA Routine für Akku in bel. Bank
B3F3:  20 E1 FF   JSR  $FFE1     Kernal STOP: Auf Stop Taste prüfen
B3F6:  F0 08      BEQ  $B400     Wenn gedrückt, dann E-Warteschleife
B3F8:  20 50 B9   JSR  $B950     Addition: Konstante <1> zum OP3
B3FB:  20 3C B9   JSR  $B93C     Subtraktion: OP2 - Konstante <1>
B3FE:  B0 EE      BCS  $B3EE     Schleifen, bis alle Steps erledigt
B400:  4C 8B B0   JMP  $B08B     Springe in Eingabe-Warteschleife
B403:  4C BC B0   JMP  $B0BC     <?> ausgeben und zur E-Warteschleife
```

```
******************************      Monitor Befehl : A  (Assemble)
                                    Monitor Befehl : .  (Assemble)

B406:  B0 3A     BCS  $B442     Carry gesetzt = KZ für Fehlerausgabe
B408:  20 01 B9  JSR  $B901     Kopieren des OP1 Inhalts nach OP3
B40B:  A2 00     LDX  # $00     Displ. auf Mnemonic-Buffer löschen
B40D:  8E A1 0A  STX  $0AA1     Bit 0 f. komprimierten Bef-Code 0
B410:  8E B4 0A  STX  $0AB4     Schleifenhilfszähler auf 0 setzen
B413:  20 E9 B8  JSR  $B8E9     Teste E-Buffer auf Bef-End, <:>, <?>
B416:  D0 07     BNE  $B41F     Wenn kein Befehlsende, dann weiter
B418:  E0 00     CPX  # $00     Displ. noch 0, dann kein Befehl da
B41A:  D0 03     BNE  $B41F     Nein, dann weiter
B41C:  4C 8B B0  JMP  $B08B     Springe in Eingabe-Warteschleife
B41F:  C9 20     CMP  # $20     Ist gelesenes Zeichen ein <Blank> ?
B421:  F0 E8     BEQ  $B40B     Ja, dann überlesen und neu inital.
B423:  9D AC 0A  STA  $0AAC,X   Zeichen in Mnemonic-Buffer ablegen
B426:  E8        INX            Displ. Zeiger auf Mnemonic-Buffer +1
B427:  E0 03     CPX  # $03     Schon 3 Mnemonic-Zeichen eingegeben?
B429:  D0 E8     BNE  $B413     Nein, dann nächstes Zeichen holen
B42B:  CA        DEX            Displ. Zeiger auf letzten Buchstaben
B42C:  30 17     BMI  $B445     3 Zeichen verarbeitet, dann weiter
B42E:  BD AC 0A  LDA  $0AAC,X   3 Mnemonic-Zeichen rückwärts lesen
B431:  38        SEC            Carry für Subtraktion setzen
B432:  E9 3F     SBC  # $3F     Alphabetwert e. Buchstabens A=1,B=2
B434:  A0 05     LDY  # $05     Zähler für 5 mal 1 Bit verschieben
B436:  4A        LSR  A         1 Bit des Buchstabenwertes aus dem
B437:  6E A1 0A  ROR  $0AA1     Akku in das Bytepaar $AA1-$AA0
B43A:  6E A0 0A  ROR  $0AA0     schieben. Die drei Mnemonic-Zeichen
B43D:  88        DEY            werden so in das o.g. Bytepaar
B43E:  D0 F6     BNE  $B436     geschoben und belegen dann 3 mal 5
B440:  F0 E9     BEQ  $B42B     Bit dieses 2-Byte-Wertes (Bit 0 frei)
B442:  4C BC B0  JMP  $B0BC     <?> ausgeben und zur E-Warteschleife
B445:  A2 02     LDX  # $02     Displacement f. Ausgabepuffer setzen
B447:  AD B4 0A  LDA  $0AB4     Schleifenhilfszähler in Akku laden
B44A:  D0 30     BNE  $B47C     Wenn ungleich Null, dann Skip
B44C:  20 CE B7  JSR  $B7CE     Hole Befehlsparameter in OP1
B44F:  F0 29     BEQ  $B47A     Wenn Null, dann teste auf Bef-End
B451:  B0 EF     BCS  $B442     Carry gesetzt = KZ für Fehlerausgabe
B453:  A9 24     LDA  # $24     Zeichen für <$> in Akku laden
B455:  9D A0 0A  STA  $0AA0,X   und in Ausgabepuffer bringen
B458:  E8        INX            Displacement auf Ausgabepuffer + 1
B459:  A5 62     LDA  * $62     Hole Bank-Byte des Operanden OP1
B45B:  D0 E5     BNE  $B442     Nicht Null = KZ für Fehlerausgabe
B45D:  A0 04     LDY  # $04     Divisionsfaktor f. Zahlenbasis 16
B45F:  AD B6 0A  LDA  $0AB6     Hole Zahlenbasis  des Operanden
B462:  C9 08     CMP  # $08     Vergleiche mit <8>
B464:  90 05     BCC  $B46B     Kleiner 8, dann hole Adr-Ho (OP1)
B466:  CC B4 0A  CPY  $0AB4     Vergleiche mit Schleifenzähler
```

```
B469:  F0 06     BEQ  $B471     Gleich, dann Skip
B46B:  A5 61     LDA  * $61     Adr-Hi Byte des OP1-Operanden holen
B46D:  D0 02     BNE  $B471     Ungleich 0, dann Skip
B46F:  A0 02     LDY  # $02     Schleifenzähler für Nullbytes setzen
B471:  A9 30     LDA  # $30     ASCII Zeichen für <0> in Akku
B473:  9D A0 0A  STA  $0AA0,X   und in den Ass.-Bef-Zwischenspeicher
B476:  E8        INX            Ass.-Bef-Längenzähler erhöhen
B477:  88        DEY            Schleifenzähler für OP-Nullbytes -1
B478:  D0 F9     BNE  $B473     Schleifen bis Zähler =0 ist
B47A:  C6 7A     DEC  * $7A     Displ. Zeiger auf vorheriges Zeichen
B47C:  20 E9 B8  JSR  $B8E9     Teste E-Buffer auf Bef-End, <:>, <?>
B47F:  F0 0E     BEQ  $B48F     Wenn Bef-End, dann zur Auswertung
B481:  C9 20     CMP  # $20     War gelesenes Zeichen ein <Blank>?
B483:  F0 C2     BEQ  $B447     Ja, dann neue Parameter-Auswertung
B485:  9D A0 0A  STA  $0AA0,X   In Ass.-Bef-Zwischenspeicher
B488:  E8        INX            Ass.-Bef-Längenzähler erhöhen
B489:  E0 0A     CPX  # $0A     Ist der Ass.-Bef länger 9 Zeichen?
B48B:  90 BA     BCC  $B447     Nein, dann nächstes Zeichen holen
B48D:  B0 B3     BCS  $B442     Ja, dann <?> Fehler ausgeben
B48F:  86 63     STX  * $63     Byte Länge d. Ass-Bef in OP2 Adr-Lo
B491:  A2 00     LDX  # $00     X-Reg mit 0 laden u. in den Befehls-
B493:  8E B1 0A  STX  $0AB1     vergleich-Schleifenzähler bringen
B496:  A2 00     LDX  # $00     X-Reg mit 0 laden und als Displ. für
B498:  86 9F     STX  * $9F     d. Ass.-Bef-Buffer benutzen
B49A:  AD B1 0A  LDA  $0AB1     Bef-Vgl-Schleifenzähler holen
B49D:  20 59 B6  JSR  $B659     Adressierung + Länge für Vgl.-Zähler
B4A0:  AE AA 0A  LDX  $0AAA     Bef-Längenzeiger holen (0,1,2)
B4A3:  86 64     STX  * $64     und in Adr-Hi (OP2) sichern
B4A5:  AA        TAX            Prüfergebnis a.Displ. f.Mnemonic-Vgl
B4A6:  BD 61 B7  LDA  $B761,X   Byte a. Mnemonic-Schlüsselwort Tab 2
B4A9:  20 7F B5  JSR  $B57F     Vergleiche m. Byte a. Ass-Bef Buffer
B4AC:  BD 21 B7  LDA  $B721,X   Byte a. Mnemonic-Schlüsselwort Tab 1
B4AF:  20 7F B5  JSR  $B57F     Vergleiche m. Byte a. Ass-Bef Buffer
B4B2:  A2 06     LDX  # $06     Schleifenzähler f. Adressierungsvgl.
B4B4:  E0 03     CPX  # $03     Wird 3. Schleife abgearbeitet?
B4B6:  D0 14     BNE  $B4CC     Nein,dann nur Adressierungsvergleich
B4B8:  AC AB 0A  LDY  $0AAB     Bef-Längenzeiger holen (0,1,2)
B4BB:  F0 0F     BEQ  $B4CC     Handelt es sich um einen 1 Byte Bef.
B4BD:  AD AA 0A  LDA  $0AAA     Adressierungsschlüssel holen
B4C0:  C9 E8     CMP  # $E8     Vergleiche mit $E8
B4C2:  A9 30     LDA  # $30     ASCII-Zeichen für <0> in Akku
B4C4:  B0 1E     BCS  $B4E4     Wenn Carry gesetzt, in entsp. Ausw.
B4C6:  20 7C B5  JSR  $B57C     Vergleiche m. Byte a. Ass-Bef Buffer
B4C9:  88        DEY            Bef-Längenzähler um 1 vermindern
B4CA:  D0 F1     BNE  $B4BD     Nicht Null, dann weiter schleifen
B4CC:  0E AA 0A  ASL  $0AAA     Adressierungsschlüssel schieben
B4CF:  90 0E     BCC  $B4DF     Wenn Bit =0, dann Skip-Vergleich
B4D1:  BD 14 B7  LDA  $B714,X   Adressierungszeichen 1 a. Tab holen
```

```
B4D4:  20 7F B5   JSR  $B57F     Vergleiche m. Byte a. Ass-Bef-Buffer
B4D7:  BD 1A B7   LDA  $B71A,X   Adressierungszeichen 2 a. Tab holen
B4DA:  F0 03      BEQ  $B4DF     Wenn $00, dann nicht vergleichen
B4DC:  20 7F B5   JSR  $B57F     Vergleiche m. Byte a. Ass-Bef Buffer
B4DF:  CA         DEX            Adressierungs-Schleifenzähler - 1
B4E0:  D0 D2      BNE  $B4B4     Nicht Null, dann weiter schleifen
B4E2:  F0 06      BEQ  $B4EA     Bei Null weiter auswerten
B4E4:  20 7C B5   JSR  $B57C     Vergleiche m. Byte a. Ass-Bef-Buffer
B4E7:  20 7C B5   JSR  $B57C     Vergleiche m. Byte a. Ass-Bef-Buffer
B4EA:  A5 63      LDA  * $63     Hole gesicherte Länge d. Ass-Bef
B4EC:  C5 9F      CMP  * $9F     Vergleiche m. Displ. a. Ass-Bef-Buf
B4EE:  F0 03      BEQ  $B4F3     Wenn gleich, dann Skip
B4F0:  4C 8B B5   JMP  $B58B     Bef-Vgl-Schleifenzähler erhöhen
B4F3:  AC AB 0A   LDY  $0AAB     Hole Befehls-Längenzeiger
B4F6:  F0 32      BEQ  $B52A     Wenn 0,dann ist es ein 1-Byte-Befehl
B4F8:  A5 64      LDA  * $64     Hole Adr-Hi-Byte des OP2 Operanden
B4FA:  C9 9D      CMP  # $9D     und vergleiche es mit $9D
B4FC:  D0 23      BNE  $B521     Nicht gleich, dann Skip
B4FE:  A5 60      LDA  * $60     Hole Operandenadresse Lo und
B500:  E5 66      SBC  * $66     subtrahiere Befehlsadresse Lo
B502:  AA         TAX            Ergebnis der Subtr. in X-Reg sichern
B503:  A5 61      LDA  * $61     Hole Operandenadresse Hi und
B505:  E5 67      SBC  * $67     subtrahiere Befehlsadresse Hi
B507:  90 08      BCC  $B511     Z. Auswertung e. Rückwärts Verzweig.
B509:  D0 6E      BNE  $B579     "Branch out of Range":<?> ausgeben
B50B:  E0 82      CPX  # $82     Prüfe, ob Branch plausibel
B50D:  B0 6A      BCS  $B579     Offset größer $82, dann <?> ausgeben
B50F:  90 08      BCC  $B519     In die entsprechende Auswertung
B511:  A8         TAY            Kopiere den Akku-Inhalt in Y-Reg
B512:  C8         INY            und erhöhe zum Testen auf Null um 1
B513:  D0 64      BNE  $B579     Nicht gleich 0, <?> Fehler ausgeben
B515:  E0 82      CPX  # $82     Vergleiche auf $02
B517:  90 60      BCC  $B579     Kleiner 2, dann <?> Fehler ausgeben
B519:  CA         DEX            Adreßabgleich:X-Reg um 1 vermindern
B51A:  CA         DEX            Adreßabgleich:X-Reg um 1 vermindern
B51B:  8A         TXA            Den Wert dann in Akku bringen
B51C:  AC AB 0A   LDY  $0AAB     Hole Befehls-Längenzähler in Y-Reg
B51F:  D0 03      BNE  $B524     Nicht Null, dann Skip
B521:  B9 5F 00   LDA  $005F,Y   Hole Wert aus Operanden OP1
B524:  20 2A B1   JSR  $B12A     STA Routine für Akku in bel. Bank
B527:  88         DEY            Befehls-Längenzähler um 1 vermindern
B528:  D0 F7      BNE  $B521     Nicht Null, dann weiter schleifen
B52A:  AD B1 0A   LDA  $0AB1     Hole den ermittelten Opcode Wert
B52D:  20 2A B1   JSR  $B12A     STA-Routine für Akku in  bel. Bank
B530:  20 AD B8   JSR  $B8AD     <Cr> <Crsr-Up> ausgeben
B533:  20 7D FF   JSR  $FF7D     Kernal PRIMM: Zeichenkette ausgeben
```

```
******************************       Monitor Konstante: Assemble Ausgabe

B536:  41 20 1B 51 00                A  <Blank>  <Esc - Q>

******************************       Kennzeichen und Adreßvorgabe für
                                     nächste Ass.-Vorgabe bilden

B53B:  20 DC B5   JSR  $B5DC         Adreßwert ausgeben und Byte holen
B53E:  EE AB 0A   INC  $0AAB         Opcode-Längenzeiger um 1 erhöhen
B541:  AD AB 0A   LDA  $0AAB         Länge zum 'Von' Operanden addieren
B544:  20 52 B9   JSR  $B952         Addition: Akku-Inhalt zum OP3
B547:  A9 41      LDA  # $41         Lade Akku mit Zeichen <A> (Assemble)
B549:  8D 4A 03   STA  $034A         In Vorgabe-Puffer für nächste Zeile
B54C:  A9 20      LDA  # $20         Lade Akku mit Zeichen <Blank>
B54E:  8D 4B 03   STA  $034B         In Vorgabe-Puffer für nächste Zeile
B551:  8D 51 03   STA  $0351         In Vorgabe-Puffer für nächste Zeile
B554:  A5 68      LDA  * $68         Bank Byte der 'Von' Adresse in Akku
B556:  20 D2 B8   JSR  $B8D2         Akku in 2-Byte-ASCII Code: Hi=A,Lo=X
B559:  8E 4C 03   STX  $034C         In Vorgabe-Puffer für nächste Zeile
B55C:  A5 67      LDA  * $67         Adr-Hi Byte (OP3) der 'Von' Adresse
B55E:  20 D2 B8   JSR  $B8D2         Akku in 2-Byte-ASCII Code: Hi=A,Lo=X
B561:  8D 4D 03   STA  $034D         In Vorgabe-Puffer für nächste Zeile
B564:  8E 4E 03   STX  $034E         In Vorgabe-Puffer für nächste Zeile
B567:  A5 66      LDA  * $66         Adr-Lo Byte (OP3) der 'Von' Adresse
B569:  20 D2 B8   JSR  $B8D2         Akku in 2 Byte ASCII Code: Hi=A,Lo=X
B56C:  8D 4F 03   STA  $034F         In Vorgabe-Puffer für nächste Zeile
B56F:  8E 50 03   STX  $0350         In Vorgabe-Puffer für nächste Zeile
B572:  A9 08      LDA  # $08         Tastaturpuffer-Zeiger auf 8 Zeichen
B574:  85 D0      STA  * $D0         hochsetzen (=Länge der Vorgabezeile)
B576:  4C 8B B0   JMP  $B08B         Springe in Eingabe Warteschleife
B579:  4C BC B0   JMP  $B0BC         <?> ausgeben und zur E-Warteschleife

******************************       Vergleiche den Akku Inhalt mit einem
                                     Zeichen aus dem Assembler-Befehl-
                                     Zwischenspeicher

B57C:  20 7F B5   JSR  $B57F         Folgende Routine 2mal ausführen
B57F:  8E AF 0A   STX  $0AAF         X-Reg-Inhalt sichern
B582:  A6 9F      LDX  * $9F         Ass-Befehl Disp.-Zeiger laden
B584:  DD A0 0A   CMP  $0AA0,X       Vgl mit Zeichen aus Ass-Bef Buffer
B587:  F0 0A      BEQ  $B593         Wenn gleich, dann Exit Routine
B589:  68         PLA                Die RTS-Adresse von Stack holen
B58A:  68         PLA                Die RTS-Adresse vom Stack holen
B58B:  EE B1 0A   INC  $0AB1         Bef-Vgl-Schleifenzähler erhöhen
B58E:  F0 E9      BEQ  $B579         Größer 255, dann Fehler ausgeben
B590:  4C 96 B4   JMP  $B496         Springe in entsp. Auswertung
B593:  E6 9F      INC  * $9F         Ass-Befehl Disp. Zeiger + 1
B595:  AE AF 0A   LDX  $0AAF         Alten X-Reg-Inhalt zurückholen
```

```
B598:  60          RTS              Rücksprung aus dem Unterprogramm

******************************      Monitor Befehl : D  (Disassemble)

B599:  B0 08       BCS  $B5A3       Kein gültiger 'Von' Operand
B59B:  20 01 B9    JSR  $B901       Kopieren des OP1 Inhalts nach OP3
B59E:  20 A7 B7    JSR  $B7A7       'Bis' Operand in OP1 holen
B5A1:  90 06       BCC  $B5A9       Wenn gültig, dann Stepzahl ermitteln
B5A3:  A9 14       LDA  # $14       Standard-Stepzahl $14 (=20 Bytes zu
B5A5:  85 60       STA  * $60       disassemblieren) in Stepzähler Lo
B5A7:  D0 05       BNE  $B5AE       Unbedingter Sprung z. Disass-Routine
B5A9:  20 0E B9    JSR  $B90E       Differenz: OP1-OP3 in OP1 speichern
B5AC:  90 23       BCC  $B5D1       Carry gelöscht = KZ f. Fehlerausgabe
B5AE:  20 7D FF    JSR  $FF7D       Kernal PRIMM: Zeichenkette ausgeben

******************************      Monitor Konstante: 1 Zeile löschen

B5B1:  0D 1B 51 00                  <Cr>  <Esc - Q>

******************************      Speicher abhängig von 'Von' Operand
                                    und Stepzahl disassemblieren

B5B5:  20 E1 FF    JSR  $FFE1       Kernal STOP: Auf Stop-Taste prüfen
B5B8:  F0 14       BEQ  $B5CE       Wenn gedrückt, in E-Warteschleife
B5BA:  20 D4 B5    JSR  $B5D4       Aufbereiten+ausgeben e.Disass. Zeile
B5BD:  EE AB 0A    INC  $0AAB       Opcode Längenzeiger um 1 erhöhen
B5C0:  AD AB 0A    LDA  $0AAB       und f.'Von' Adreßberechnung in Akku
B5C3:  20 52 B9    JSR  $B952       Addition: Akku Inhalt zum OP3
B5C6:  AD AB 0A    LDA  $0AAB       Längenz. f. Stepzahl-Berechnung in A
B5C9:  20 24 B9    JSR  $B924       Subtraktion: OP1 - Akku Inhalt
B5CC:  B0 E0       BCS  $B5AE       Noch Steps offen,dann weiter disass.
B5CE:  4C 8B B0    JMP  $B08B       Springe in Eingabe-Warteschleife
B5D1:  4C BC B0    JMP  $B0BC       <?> Ausgeben und zur E-Warteschleife
B5D4:  A9 2E       LDA  # $2E       Akku mit <.> laden
B5D6:  20 D2 FF    JSR  $FFD2       Kernal BSOUT: Ein Zeichen ausgeben
B5D9:  20 A8 B8    JSR  $B8A8       <Blank>, <Cr>, <Crsr-Up> ausgeben
B5DC:  20 92 B8    JSR  $B892       'Von' Adr (OP3) a.5 Byte ASCII ausg.
B5DF:  20 A8 B8    JSR  $B8A8       <Blank>, <Cr>, <Crsr-Up> ausgeben
B5E2:  A0 00       LDY  # $00       Displacement für FETCH Routine laden
B5E4:  20 1A B1    JSR  $B11A       LDA Routine für Akku aus bel. Bank
B5E7:  20 59 B6    JSR  $B659       Gültigkeitsprüfung des Opcode-Bytes
B5EA:  48          PHA              Sichere Prüfungsergebnis auf Stack
B5EB:  AE AB 0A    LDX  $0AAB       Hole ermittelten Befehls-Längenschl.
B5EE:  E8          INX              Längenschlüssel um 1 erhöhen
B5EF:  CA          DEX              Längenschlüssel um 1 vermindern
B5F0:  10 0A       BPL  $B5FC       B. Wert kleiner 0 Konstante ausgeben
B5F2:  20 7D FF    JSR  $FF7D       Kernal PRIMM: Zeichenkette ausgeben
```

```
******************************     Monitor Konstante: 3 Leerzeichen

B5F5:  20 20 20 00                 <Blank>  <Blank>  <Blank>

******************************     Ass. Disass. Unterroutine

B5F9:  4C 02 B6   JMP  $B602       Skip LDA für Routine
B5FC:  20 1A B1   JSR  $B11A       LDA-Routine für Akku aus bel. Bank
B5FF:  20 A5 B8   JSR  $B8A5       Akku als 2-Byte-ASCII +Blank ausgeb.
B602:  C8         INY              Inhalt Y-Reg um 1 erhöhen
B603:  C0 03      CPY  # $03       Vergleiche auf $03
B605:  90 E8      BCC  $B5EF       Kleiner 3, dann weiter schleifen
B607:  68         PLA              Prüfungsergebnis vom Stack holen
B608:  A2 03      LDX  # $03       3 Buchstaben f. Mnemonic Ausgabe in
B60A:  20 A1 B6   JSR  $B6A1       X-Reg u. zur Ausgabe der Buchstaben
B60D:  A2 06      LDX  # $06       Schleifenzähler mit 6 initialisieren
B60F:  E0 03      CPX  # $03       Nach 3 Schleifen wird d. eigentliche
B611:  D0 17      BNE  $B62A       Adreßwert ausgegeben
B613:  AC AB 0A   LDY  $0AAB       Anzahl der Befehls Operandenbytes
B616:  F0 12      BEQ  $B62A       Kein Operandenbyte, dann Skip
B618:  AD AA 0A   LDA  $0AAA       Adressierungsschlüssel d. Befehls
B61B:  C9 E8      CMP  # $E8       Prüfe, ob es ein Branch ist
B61D:  08         PHP              Carry Flag auf Stack sichern
B61E:  20 1A B1   JSR  $B11A       LDA Routine für Akku aus bel. Bank
B621:  28         PLP              Carry Flag wieder zurückholen
B622:  B0 1D      BCS  $B641       War Carry gesetzt,dann ist es BRANCH
B624:  20 C2 B8   JSR  $B8C2       Akku in 2-Byte-ASCII ausgeben
B627:  88         DEY              Wenn der Befehl zwei Operandenbytes
B628:  D0 EE      BNE  $B618       hat, dann zur Auswertung des zweiten
B62A:  0E AA 0A   ASL  $0AAA       Bit aus Adressierungsschlüssel mask.
B62D:  90 0E      BCC  $B63D       Bit nicht gesetzt, dann Skip
B62F:  BD 14 B7   LDA  $B714,X     1 Zeichen für Adressierungsart holen
B632:  20 D2 FF   JSR  $FFD2       Kernal BSOUT: Ein Zeichen ausgeben
B635:  BD 1A B7   LDA  $B71A,X     1 Zeichen für Adressierungsart holen
B638:  F0 03      BEQ  $B63D       ungleich $00, dann ausgeben
B63A:  20 D2 FF   JSR  $FFD2       Kernal BSOUT: Ein Zeichen ausgeben
B63D:  CA         DEX              Schleifenzähler-Adreßausgabe - 1
B63E:  D0 CF      BNE  $B60F       Alle 6 Schleifen ausführen
B640:  60         RTS              Rücksprung aus dem Unterprogramm

******************************     Adresse v. BRANCH Befehlen ermitteln

B641:  20 4D B6   JSR  $B64D       Zur Adreßberechnung  X=Hi, A=Lo
B644:  18         CLC              Carry für Addition löschen
B645:  69 01      ADC  # $01       1 für Adreßkorrektur Lo addieren
B647:  D0 01      BNE  $B64A       Kein Überlauf, Skip-Hi-Korrektur
B649:  E8         INX              1 für Adreßkorrektur Hi addieren
B64A:  4C 9F B8   JMP  $B89F       Akku +X-Reg als 4-Byte-ASCII ausgeb.
```

```
B64D:  A6 67      LDX  * $67      Adr-Hi d. 'Von' Operanden OP3 holen
B64F:  A8         TAY             Den Branch-Offset in Y-Reg bringen
B650:  10 01      BPL  $B653      Weiter bei Branch "vorwärts"
B652:  CA         DEX             Adr-Hi für "rückwärts" Branch -1
B653:  65 66      ADC  * $66      Adr-Lo (OP3) +Branch Offset addieren
B655:  90 01      BCC  $B658      Kein Überlauf, Skip-Hi-Korrektur
B657:  E8         INX             Überlaufkorrektur des Adr-Hi-Wertes
B658:  60         RTS             Rücksprung aus dem Unterprogramm

******************************    Adressierung und Länge für den in A
                                  übergebenen Prüfcode ermitteln

B659:  A8         TAY             Prüfcode in Y-Reg sichern
B65A:  4A         LSR  A          Bit 0 herausschieben und testen
B65B:  90 0B      BCC  $B668      War Bit 0 = 0 dann OK
B65D:  4A         LSR  A          Bit 1 herausschieben und testen
B65E:  B0 17      BCS  $B677      War Bit 1 = 1 dann ungültig
B660:  C9 22      CMP  # $22      Teste, ob Ausgangscode $89 war
B662:  F0 13      BEQ  $B677      Ja, dann in Routine für ungültig
B664:  29 07      AND  # $07      Bit 3-7 des Wertes ausmaskieren
B666:  09 80      ORA  # $80      Bit 7 einblenden
B668:  4A         LSR  A          Akku-Inhalt durch 2 dividieren
B669:  AA         TAX             und als Displ. nach X-Reg
B66A:  BD C3 B6   LDA  $B6C3,X    1 Byte a. Adressierungs-Referenz-Tab
B66D:  B0 04      BCS  $B673      Wenn Rest bei Division, dann Skip
B66F:  4A         LSR  A          Kopiere den Inhalt
B670:  4A         LSR  A          der oberen Tetrade
B671:  4A         LSR  A          (Bit 4-7) in die
B672:  4A         LSR  A          untere Tetrade (Bit 0-3)
B673:  29 0F      AND  # $0F      Obere Tetrade (Bit 4-7) ausmaskieren
B675:  D0 04      BNE  $B67B      Ungleich 0 ist gültig
B677:  A0 80      LDY  # $80      Bei ungültigem Code lade Y mit $80
B679:  A9 00      LDA  # $00      und lade Akku mit $00
B67B:  AA         TAX             A f. Displacement n. X transferieren
B67C:  BD 07 B7   LDA  $B707,X    Hole Adressierungsschlüssel aus Tab
B67F:  8D AA 0A   STA  $0AAA      und sichere ihn in $0AAA
B682:  29 03      AND  # $03      Ausmaskieren der Bits 2-7
B684:  8D AB 0A   STA  $0AAB      Sichere Bit 0,1 (Befehlslänge)
B687:  98         TYA             Prüfcode in Akku kopieren
B688:  29 8F      AND  # $8F      Die Bits 4,5,6 ausmaskieren
B68A:  AA         TAX             Den ausmaskierten Wert in X sichern
B68B:  98         TYA             Prüfcode in Akku kopieren
B68C:  A0 03      LDY  # $03      Schleifenzähler mit 3 initialisieren
B68E:  E0 8A      CPX  # $8A      Vgl. ausmaskierten Wert mit $8A
B690:  F0 0B      BEQ  $B69D      Gleich, dann Skip
B692:  4A         LSR  A          Dividiere-Akku-Inhalt durch <2>
B693:  90 08      BCC  $B69D      Wenn kein Rest, dann Skip
B695:  4A         LSR  A          Dividiere-Akku-Inhalt durch <2>
```

```
B696:  4A        LSR  A          Dividiere-Akku-Inhalt durch <2>
B697:  09 20     ORA  # $20      Bit 5 in Akku einblenden
B699:  88        DEY             Schleifenzähler um 1 vermindern
B69A:  D0 FA     BNE  $B696      Nicht Null, dann weiter schleifen
B69C:  C8        INY             Schleifenzähler um 1 erhöhen
B69D:  88        DEY             Schleifenzähler um 1 vermindern
B69E:  D0 F2     BNE  $B692      Nicht Null, dann weiter dividieren
B6A0:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   Einen Buchstaben für Mnemonic-
                                 Anzeige aufbereiten und ausgeben

B6A1:  A8        TAY             Befehls-Code als Displ. nach Y-Reg
B6A2:  B9 21 B7  LDA  $B721,Y    Ein Byte aus Mnemonic-Tabelle 1
B6A5:  85 63     STA  * $63      holen und in Adr-Lo (OP2) sichern
B6A7:  B9 61 B7  LDA  $B761,Y    Ein Byte aus Mnemonic-Tabelle 2
B6AA:  85 64     STA  * $64      holen und in Adr-Hi (OP2) sichern
B6AC:  A9 00     LDA  # $00      Akku mit 0 initialisieren
B6AE:  A0 05     LDY  # $05      5 Bits aus der 2-Byte-Adresse des
B6B0:  06 64     ASL  * $64      OP2-Operanden nach links heraus-
B6B2:  26 63     ROL  * $63      schieben und diese 5 Bits im Akku
B6B4:  2A        ROL  A          ablegen. Schleifen, bis alle 5 Bits
B6B5:  88        DEY             herausgeschoben sind. Die Addition
B6B6:  D0 F8     BNE  $B6B0      von $3F ergibt dann einen gültigen
B6B8:  69 3F     ADC  # $3F      Buchstabenwert, bzw. ein <?>
B6BA:  20 D2 FF  JSR  $FFD2      Kernal BSOUT: Ein Zeichen ausgeben
B6BD:  CA        DEX             3 Schleifen, f. die 3 Buchstaben aus
B6BE:  D0 EC     BNE  $B6AC      dem 16-Bit-Wert in Adr-Lo/Hi in OP2
B6C0:  4C A8 B8  JMP  $B8A8      Ausgeben:<Blank>,<Cr>,<Crsr-Up>:RTS

******************************   Adressierungsreferenz-Tabelle

B6C3:  40 02 45 03 D0 08 40 09
B6CB:  30 22 45 33 D0 08 40 09
B6D3:  40 02 45 33 D0 08 40 09
B6DB:  40 02 45 B3 D0 08 40 09
B6E3:  00 22 44 33 D0 8C 44 00
B6EB:  11 22 44 33 D0 8C 44 9A
B6F3:  10 22 44 33 D0 08 40 09
B6FB:  10 22 44 33 D0 08 40 09
B703:  62 13 78 A9
```

```
******************************     Adressierungsart und Längenschlüssel

B707:  00 21 81 82                    -1 / #$    -2 / *$    -2 / $     -3
B70B:  00 00 59 4D                    -1 /       -1 / ($,X)-2 / ($),Y-2
B70F:  91 92 86 4A                 *$,X-2 / $,X  -3 / $,Y  -3 / ($)  -3
B713:  85                          *$,Y-2

B714:  9D          .Byte  $9D      Backspace Steuercode

******************************     Adressierungsarten optisch anzeigen

B715:  2C 29 2C 23 28 24           < , > < ) > < , > < # > < ( > < $ >
B71B:  59 00 58 24 24 00           < Y > <   > < X > < $ > < $ > <   >

******************************     Mnemonic-Schlüsselwort Tabelle 1

B721:  1C 8A 1C 23 5D 8B 1B A1     BRK PHP BPL CLC JSR PLP BMI SEC
B729:  9D 8A 1D 23 9D 8B 1D A1     RTI PHA BVC CLI RTS PLA BVS SEI
B731:  00 29 19 AE 69 A8 19 23     ??? DEY BCC TYA LDY TAY BCS CLV
B739:  24 53 1B 23 24 53 19 A1     CPY INY BNE CLD CPX INX BEQ SED
B741:  00 1A 5B 5B A5 69 24 24     ??? BIT JMP JMP STY LDY CPY CPX
B749:  AE AE A8 AD 29 00 7C 00     TXA TXS TAX TSX DEX ??? NOP ???
B751:  15 9C 6D 9C A5 69 29 53     ASL ROL LSR ROR STX LDX DEC INC
B759:  84 13 34 11 A5 69 23 A0     ORA AND EOR ADC STA LDA CMP SBC

******************************     Mnemonic-Schlüsselwort Tabelle 2

B761:  D8 62 5A 48 26 62 94 88     Jeweils 1 Byte aus der Tabelle 1
B769:  54 44 C8 54 68 44 E8 94     ergibt mit dem korrespondierenden
B771:  00 B4 08 84 74 B4 28 6E     Wert aus d. Tabelle 2 ein. 16-Bit
B779:  74 F4 CC 4A 72 F2 A4 8A     Wert, in dem die 3 Buchstaben f. d.
B781:  00 AA A2 A2 74 74 74 72     Mnemonics codiert sind. Das 16-Bit
B789:  44 68 B2 32 B2 00 22 00     Argument wird in 3 Teile zu je 5
B791:  1A 1A 26 26 72 72 88 C8     Bits aufgeteilt. Je ein Teil codiert
B799:  C4 CA 26 48 44 44 A2 C8     einen Buchstaben. Bit 0 = ungenutzt.

******************************     Monitor Konstante: 3 Leerzeichen

B7A1:  0D 20 20 20                 <Cr>  <Blank>  <Blank>  <Blank>

******************************     Prüfe auf gültiges Trennzeichen
                                   zwischen den Befehlsoperanden

B7A5:  C6 7A       DEC  * $7A      E-Buf.-Zeiger auf vorheriges Zeichen
B7A7:  20 CE B7    JSR  $B7CE      Operanden in OP1 holen
B7AA:  B0 16       BCS  $B7C2      Carry gesetzt = KZ für Fehler
B7AC:  20 E7 B8    JSR  $B8E7      Letztes Zeichen erneut lesen
B7AF:  D0 09       BNE  $B7BA      War es kein Bef-End, dann weiter
```

```
B7B1:  C6 7A     DEC  * $7A     E-Buf.-Zeiger auf vorheriges Zeichen
B7B3:  AD B4 0A  LDA  $0AB4     Hilfsflag für Fehlererkennung holen
B7B6:  D0 11     BNE  $B7C9     Nicht Null, dann OK Exit
B7B8:  F0 0D     BEQ  $B7C7     Kein gültiger Operand, Fehler Exit
B7BA:  C9 20     CMP  # $20     War gelesenes Zeichen ein <Blank> ?
B7BC:  F0 0B     BEQ  $B7C9     Gültiges Trennzeichen, OK Exit
B7BE:  C9 2C     CMP  # $2C     War gelesenes Zeichen ein <Komma> ?
B7C0:  F0 07     BEQ  $B7C9     Gültiges Trennzeichen, OK Exit
B7C2:  68        PLA            Die auf dem Stack abgelegte Adresse
B7C3:  68        PLA            d. Befehls wieder löschen, da Fehler
B7C4:  4C BC B0  JMP  $B0BC     <?> Ausgeben und zur E-Warteschleife

*******************************  Exit für Fehler in Befehlsoperanden

B7C7:  38        SEC            Carry setzen = Fehlerkennzeichen
B7C8:  24        .Byte  $24     Skip nach $B7CA

*******************************  Befehlsoperand/Trennzeichen ist OK

B7C9:  18        CLC            Carry gelöscht = Kennzeichen für OK
B7CA:  AD B4 0A  LDA  $0AB4     Hilfsflag für Fehlererkennung laden
B7CD:  60        RTS            Schein RTS ist der Befehlseinsprung

*******************************  Initialisierung und Auswertung eines
                                 Befehlsparameters in OP1

B7CE:  A9 00     LDA  # $00     Für Parameterinit. Akku mit 0 laden
B7D0:  85 60     STA  * $60     Löschen des 3-Byte-Befehlsparameters
B7D2:  85 61     STA  * $61     Nr. 1 (OP1), der i.d. Zero-Page von
B7D4:  85 62     STA  * $62     $62 (Highest) - $60 (Lowest) steht
B7D6:  8D B4 0A  STA  $0AB4     Hilfsspeicher für Fehlerkontrolle
B7D9:  8A        TXA            X-Reg in Akku bringen
B7DA:  48        PHA            und auf Stack retten
B7DB:  98        TYA            Y-Reg in Akku bringen
B7DC:  48        PHA            und auf Stack retten
B7DD:  20 E9 B8  JSR  $B8E9     Teste E-Buffer auf Bef-End, <:>, <?>
B7E0:  D0 03     BNE  $B7E5     Kein Ende-KZ, dann weiter machen
B7E2:  4C 7E B8  JMP  $B87E     Exit-Routine mit Carry-Clear-Kennz.
B7E5:  C9 20     CMP  # $20     War Zeichen ein <Blank> ?
B7E7:  F0 F4     BEQ  $B7DD     Ja, dann nächstes Zeichen lesen
B7E9:  A2 03     LDX  # $03     Displ. f. 4 Umrechnungszeichen holen
B7EB:  DD F5 B0  CMP  $B0F5,X   Prüfen auf Umrechnungszeichen (%&+$)
B7EE:  F0 06     BEQ  $B7F6     weiter bei Umrechnungsz. gefunden
B7F0:  CA        DEX            Displ auf Umrechnungsz.-Tab  - 1
B7F1:  10 F8     BPL  $B7EB     Schleifen bis Tabelle ganz durch
B7F3:  E8        INX            X-Reg auf $0 setzen (= Basis Hex)
B7F4:  C6 7A     DEC  * $7A     Displacement Zeiger auf E-Buffer - 1
B7F6:  BC 8A B8  LDY  $B88A,X   Lade Y-Reg m. Basis d. Zahlensystems
```

```
B7F9:  BD 8E B8  LDA  $B88E,X    Lade Akku m. Multiplikationsfaktor
B7FC:  8D B6 0A  STA  $0AB6      für das Zahlensystem und sichere ihn
B7FF:  20 E9 B8  JSR  $B8E9      Teste E-Buffer auf Bef-End, <:>, <?>
B802:  F0 7A     BEQ  $B87E      Exit aus Routine Operandenermittlung
B804:  38        SEC             Carry für ordentliche Subtr. setzen
B805:  E9 30     SBC  # $30      Zeichen in Fixpunkt-Wert wandeln
B807:  90 75     BCC  $B87E      Wenn Zeichen kleiner <0> dann Exit
B809:  C9 0A     CMP  # $0A      War Zeichen Ziffer zwischen 0 - 9 ?
B80B:  90 06     BCC  $B813      Ja, dann Hex-Anpassung überspringen
B80D:  E9 07     SBC  # $07      Anpassung der Hex-Werte  A - F
B80F:  C9 10     CMP  # $10      Wenn Wert nicht zwischen 0 - F, dann
B811:  B0 6B     BCS  $B87E      Exit aus Routine Operandenermittlung
B813:  8D B5 0A  STA  $0AB5      Ermittelten gültigen Hexwert sichern
B816:  CC B5 0A  CPY  $0AB5      Vergleiche Basis mit Hexwert
B819:  90 61     BCC  $B87C      Wenn Basis "<" Zeichen, dann Fehler
B81B:  F0 5F     BEQ  $B87C      Wenn Basis "=" Zeichen, dann Fehler
B81D:  EE B4 0A  INC  $0AB4      Hilfsbyte für Fehlererkennung + 1
B820:  C0 0A     CPY  # $0A      Wurde Dezimaleingabe gewählt ?
B822:  D0 0A     BNE  $B82E      Nein, dann Überspringe Dezimalinit.
B824:  A2 02     LDX  # $02      Schleifenzähler auf 2 setzen
B826:  B5 60     LDA  * $60,X    Kopieren des 3-Byte-Operanden (OP1)
B828:  9D B7 0A  STA  $0AB7,X    in den 3-Byte-Hilfsoperanden für
B82B:  CA        DEX             dezimale Adreßeingabe
B82C:  10 F8     BPL  $B826      ($0AB9= Highest, $0AB7= Lowest)
B82E:  AE B6 0A  LDX  $0AB6      hole Zähler f. Multiplikationsfaktor
B831:  06 60     ASL  * $60      Den 3-Byte-
B833:  26 61     ROL  * $61      Operanden  (OP1)
B835:  26 62     ROL  * $62      mit <2> multiplizieren
B837:  B0 43     BCS  $B87C      Wenn Überlauf vorhanden, dann Fehler
B839:  CA        DEX             Schleifenzähler 2er Multiplik. - 1
B83A:  D0 F5     BNE  $B831      Schleifen zur OP1-Multiplikation
B83C:  C0 0A     CPY  # $0A      Ist Zahlenbasis das Dezimalsystem ?
B83E:  D0 22     BNE  $B862      Nein, Überspringe Dezimalabgleich
B840:  0E B7 0A  ASL  $0AB7      Dezimalabgleich: Den 3-Byte-
B843:  2E B8 0A  ROL  $0AB8      Hilfsoperanden in $AB9 - $AB7
B846:  2E B9 0A  ROL  $0AB9      mit <2> multiplizieren
B849:  B0 31     BCS  $B87C      Wenn Überlauf vorhanden, dann Fehler
B84B:  AD B7 0A  LDA  $0AB7      Addition des 3-Byte-
B84E:  65 60     ADC  * $60      Hilfsoperanden in den
B850:  85 60     STA  * $60      Speicherstellen
B852:  AD B8 0A  LDA  $0AB8      $0AB9-$0AB8-$0AB7
B855:  65 61     ADC  * $61      zum Inhalt des 3 Byte
B857:  85 61     STA  * $61      Operanden OP1 unter Beachtung
B859:  AD B9 0A  LDA  $0AB9      eines evtl. vorhandenen Überlaufs
B85C:  65 62     ADC  * $62      Das Ergebnis der Addition wird
B85E:  85 62     STA  * $62      im Operanden 1 (OP1) abgelegt
B860:  B0 1A     BCS  $B87C      Wenn Überlauf vorhanden, dann Fehler
B862:  18        CLC             Lösche Carry (f. Bin, Okt, Hex Syst)
```

```
B863:  AD B5 0A    LDA  $0AB5      Hole ermittelten Wert des Zeichens
B866:  65 60       ADC  * $60      Addiere den Wert auf die
B868:  85 60       STA  * $60      niederwertigste OP1-Stelle auf
B86A:  8A          TXA             Akkumulator mit Null laden
B86B:  65 61       ADC  * $61      Berücksichtige Überlauf bei Addition
B86D:  85 61       STA  * $61      auf niederwertigste OP1-Stelle
B86F:  8A          TXA             Akkumulator mit Null laden
B870:  65 62       ADC  * $62      Berücksichtige Überlauf a. mittlerer
B872:  85 62       STA  * $62      Stelle bei OP1-Addition
B874:  B0 06       BCS  $B87C      Wenn Überlauf vorhanden, dann Fehler
B876:  29 F0       AND  # $F0      Untere Tetrade (B. 0-3) ausmaskieren
B878:  D0 02       BNE  $B87C      Wenn obere Tetrade <> 0, dann Fehler
B87A:  F0 83       BEQ  $B7FF      Nächste Operandenstelle auswerten

******************************     Exit Parameterauswertung bei Fehler

B87C:  38          SEC             Carry setzen = KZ f. Fehler gefunden
B87D:  24          .Byte  $24      Skip nach $B87F

******************************     Exit Parameterauswertung bei OK

B87E:  18          CLC             Carry löschen = KZ für Parameter OK
B87F:  8C B6 0A    STY  $0AB6      sichere Basis des Zahlensystems d.OP
B882:  68          PLA             Den auf dem Stack gesicherten alten
B883:  A8          TAY             Y-Reg-Inhalt wiederherstellen
B884:  68          PLA             Den auf dem Stack gesicherten alten
B885:  AA          TAX             X-Reg Inhalt wiederherstellen
B886:  AD B4 0A    LDA  $0AB4      Akku mit Fehler-Hilfzeiger laden
B889:  60          RTS             Rücksprung aus dem Unterprogramm

******************************     Basen der 4 Zahlensysteme

B88A:  10 0A 08 02                 Hex, Dezi, Oktal, Binär

******************************     Anzahl der Multiplikationen mit dem
                                   Faktor 2 für die Zahlensysteme

B88E:  04 03 03 01                 Hex, Dezi, Oktal, Binär

******************************     OP3 Inhalt als 5-Byte-ASCII ausgeben

B892:  A5 68       LDA  * $68      Lade A mit Highest (Bank) Byte (OP3)
B894:  20 D2 B8    JSR  $B8D2      Akku in 2-Byte-ASCII-Code: Hi=A,Lo=X
B897:  8A          TXA             ASCII Code des Lo Wertes in Akku
B898:  20 D2 FF    JSR  $FFD2      Kernal BSOUT: Ein Zeichen ausgeben
B89B:  A5 66       LDA  * $66      Lade A m. Lowest (Adr-Lo) Byte (OP3)
B89D:  A6 67       LDX  * $67      Lade X m. Middle (Adr-Hi) Byte (OP3)
B89F:  48          PHA             Akku Inhalt auf Stapel sichern
```

```
B8A0:  8A        TXA              Den Adr-Hi-Wert aus OP3 in Akku
B8A1:  20 C2 B8  JSR  $B8C2       Akku in 2-Zeichen-ASCII-Code ausgeb.
B8A4:  68        PLA              Akku wieder mit Adr-Lo (OP3) laden

*******************************   Akku in ASCII aufbereiten, ausgeben,
                                  <Blank> ausgeben, zum Zeilenanfang

B8A5:  20 C2 B8  JSR  $B8C2       Akku in 2-Zeichen-ASCII-Code ausgeb.
B8A8:  A9 20     LDA  # $20       <Blank> in Akku bringen
B8AA:  4C D2 FF  JMP  $FFD2       Kernal BSOUT: Ein Zeichen ausgeben
B8AD:  20 7D FF  JSR  $FF7D       Kernal PRIMM: Zeichenkette ausgeben

*******************************   Monitor-Ausgabe-Konstante

B8B0:  0D 91 00                   <Cr> <Crsr Up>

*******************************   Ende der Ausgabe-Routine

B8B3:  60        RTS              Rücksprung aus dem Unterprogramm

*******************************   <Cr> <Cr> <Esc-Q> Blank ausgeben

B8B4:  A9 0D     LDA  # $0D       <Cr> Code in Akku laden
B8B6:  4C D2 FF  JMP  $FFD2       Kernal BSOUT: Ein Zeichen ausgeben
B8B9:  20 7D FF  JSR  $FF7D       Kernal PRIMM: Zeichenkette ausgeben

*******************************   Monitor-Konstante für Wagenrücklauf
                                  und nächste Zeile löschen, Blank aus

B8BC:  0D 1B 51 20 00             <Cr> <Esc-Q> <Blank>

*******************************   Ende der Ausgabe-Routine

B8C1:  60        RTS              Rücksprung aus dem Unterprogramm

*******************************   Akku-Inhalt in 2-Byte-Character For-
                                  mat wandeln und über BSOUT ausgeben

B8C2:  8E AF 0A  STX  $0AAF       Alten Inhalt X-Reg sichern
B8C5:  20 D2 B8  JSR  $B8D2       Akku in 2 Byte-ASCII-Code: Hi=A,Lo=X
B8C8:  20 D2 FF  JSR  $FFD2       Kernal BSOUT: Ein Zeichen ausgeben
B8CB:  8A        TXA              Character aus X in Akku laden
B8CC:  AE AF 0A  LDX  $0AAF       Alten Inhalt X-Reg wiederherstellen
B8CF:  4C D2 FF  JMP  $FFD2       Kernal BSOUT: Ein Zeichen ausgeben
```

```
*****************************       Akku Inhalt splitten und in 2 Byte
                                    ASCII Code wandeln (X = Lo, A = Hi)

B8D2:  48        PHA                Akku Inhalt zwischenspeichern
B8D3:  20 DC B8  JSR  $B8DC         Tetrade Lo in ASCII wandeln
B8D6:  AA        TAX                ASCII Code der Tetrade Lo in X-Reg
B8D7:  68        PLA                Akku Inhalt zurückholen
B8D8:  4A        LSR  A             4mal nach rechts shiften,
B8D9:  4A        LSR  A             um so die obere Tetrade (Bits 4-7)
B8DA:  4A        LSR  A             an die Position der unteren Tetrade
B8DB:  4A        LSR  A             (Bits 0-3) zu bringen

*****************************       Wert der unteren Tetrade des Akku
                                    Inhalts in ASCII Code wandeln

B8DC:  29 0F     AND  # $0F         Obere Tetrade (Bit 4-7) ausmaskieren
B8DE:  C9 0A     CMP  # $0A         Ist es eine Ziffer zwischen 0-9
B8E0:  90 02     BCC  $B8E4         Ja, dann zur Erzeugung f. ASCII Code
B8E2:  69 06     ADC  # $06         Character-Anpassung für Ziffern A-F
B8E4:  69 30     ADC  # $30         ASCII-Code für Akku Inhalt erzeugen
B8E6:  60        RTS                Rücksprung aus dem Unterprogramm

*****************************       1 Zeichen a. Eingabebuffer holen und
                                    b. BEF-End,<:>,<?> Equal Flag setzen

B8E7:  C6 7A     DEC  * $7A         Displ. auf E-Buffer - 1 (wie CHRGOT)
B8E9:  8E AF 0A  STX  $0AAF         Inhalt X-Reg sichern
B8EC:  A6 7A     LDX  * $7A         X-Reg mit Displ. auf E-Buffer laden
B8EE:  BD 00 02  LDA  $0200,X       Hole Zeichen aus Befehls E-Buffer
B8F1:  F0 06     BEQ  $B8F9         Wurde $00 = KZ für Bef-End gefunden
B8F3:  C9 3A     CMP  # $3A         Ist gelesenes Zeichen ein <:> ?
B8F5:  F0 02     BEQ  $B8F9         Ja, Exit mit gesetztem Equal Flag
B8F7:  C9 3F     CMP  # $3F         Ist gelesenes Zeichen ein <?> ?
B8F9:  08        PHP                Status des Equal Flag sichern
B8FA:  E6 7A     INC  * $7A         Displ. auf E-Buffer + 1 (next Char.)
B8FC:  AE AF 0A  LDX  $0AAF         Alten Inhalt d. X-Reg zurückholen
B8FF:  28        PLP                Status Equal Flag wiederherstellen
B900:  60        RTS                Rücksprung aus dem Unterprogramm

*****************************       Kopiere Inhalt des OP1 ($62-$61-$60)
                                    nach OP3 ($68-$67-$66)

B901:  A5 60     LDA  * $60         OP1 Lowest (Adr-Lo) holen und nach
B903:  85 66     STA  * $66         OP3 Lowest (Adr-Lo) kopieren
B905:  A5 61     LDA  * $61         OP1 Middle (Adr-Hi) holen und nach
B907:  85 67     STA  * $67         OP3 Middle (Adr-Hi) kopieren
B909:  A5 62     LDA  * $62         OP1 Highest (Bank-Byte) nach
B90B:  85 68     STA  * $68         OP3 Highest (Bank-Byte) kopieren
```

```
B90D:  60         RTS          Rücksprung aus dem Unterprogramm

******************************  Differenz: OP1-OP3 in OP1 speichern

B90E:  38         SEC          Carry für Subtraktion setzen
B90F:  A5 60      LDA  * $60   Akku mit OP1 Lowest laden
B911:  E5 66      SBC  * $66   Subtrahiere OP3 Lowest davon
B913:  85 60      STA  * $60   Speichere Ergebnis in OP1 Lowest
B915:  A5 61      LDA  * $61   Akku mit OP1 Middle laden
B917:  E5 67      SBC  * $67   Subtrahiere OP3 Middle (+ Unterlauf)
B919:  85 61      STA  * $61   Speichere Ergebnis in OP3 Middle
B91B:  A5 62      LDA  * $62   Akku mit OP1 Highest laden
B91D:  E5 68      SBC  * $68   Subtr. OP3 Highest (+ Unterlauf)
B91F:  85 62      STA  * $62   Speichere Ergebnis in OP1 Highest
B921:  60         RTS          Rücksprung aus dem Unterprogramm

******************************  Subtraktion: OP1 - Minuend in $0AAF

B922:  A9 01      LDA  # $01   Akku mit 1 laden und als
B924:  8D AF 0A   STA  $0AAF   Minuend in $0AAF speichern
B927:  38         SEC          Carry für Subtraktion setzen
B928:  A5 60      LDA  * $60   Akku mit OP1 Lowest laden
B92A:  ED AF 0A   SBC  $0AAF   Minuend von OP1 Lowest subtrahieren
B92D:  85 60      STA  * $60   Ergebnis der Subtr. zurückschreiben
B92F:  A5 61      LDA  * $61   Akku mit OP1 Middle laden
B931:  E9 00      SBC  # $00   Unterlauf d. Lowest-Subtr. beachten
B933:  85 61      STA  * $61   Ergebnis der Subtr. zurückschreiben
B935:  A5 62      LDA  * $62   Akku mit OP1 Highest laden
B937:  E9 00      SBC  # $00   Unterlauf d. Middle-Subtr. beachten
B939:  85 62      STA  * $62   Ergebnis der Subtr. zurückschreiben
B93B:  60         RTS          Rücksprung aus dem Unterprogramm

******************************  Subtraktion der Konstante 1 vom
                                Operanden 2 (OP2) in $65-$64-$63

B93C:  38         SEC          Carry für Subtraktion setzen
B93D:  A5 63      LDA  * $63   Akku mit OP2 Lowest laden
B93F:  E9 01      SBC  # $01   davon 1 subtrahieren
B941:  85 63      STA  * $63   Ergebnis der Subtr. zurückschreiben
B943:  A5 64      LDA  * $64   Akku mit OP2 Middle laden
B945:  E9 00      SBC  # $00   Unterlauf d. Lowest-Subtr. beachten
B947:  85 64      STA  * $64   Ergebnis der Subtr. zurückschreiben
B949:  A5 65      LDA  * $65   Akku mit OP3 Highest laden
B94B:  E9 00      SBC  # $00   Unterlauf d. Middle-Subtr. beachten
B94D:  85 65      STA  * $65   Ergebnis der Subtr. zurückschreiben
B94F:  60         RTS          Rücksprung aus dem Unterprogramm
```

```
*******************************    Addition des Akku-Inhalts zum OP3

B950:  A9 01     LDA  # $01        Akku mit Additionskonstante 1 laden
B952:  18        CLC               Carry für Addition löschen
B953:  65 66     ADC  * $66        Inhalt von OP3 Lowest addieren
B955:  85 66     STA  * $66        Ergebnis der Addit. zurückschreiben
B957:  90 06     BCC  $B95F        Wenn kein Überlauf, dann Rücksprung
B959:  E6 67     INC  * $67        Bei Überlauf OP3 Middle hochzählen
B95B:  D0 02     BNE  $B95F        Wenn kein Überlauf, dann Rücksprung
B95D:  E6 68     INC  * $68        Bei Überlauf OP3 Highest hochzählen
B95F:  60        RTS               Rücksprung aus dem Unterprogramm

*******************************    Subtraktion der Konstante 1 von OP3

B960:  38        SEC               Carry für Subtraktion setzen
B961:  A5 66     LDA  * $66        Akku mit OP3 Lowest laden
B963:  E9 01     SBC  # $01        Konstante 1 davon abziehen
B965:  85 66     STA  * $66        Ergebnis der Subtr. zurückschreiben
B967:  A5 67     LDA  * $67        Akku mit OP3 Middle laden
B969:  E9 00     SBC  # $00        Unterlauf der Lowest Subtr. beachten
B96B:  85 67     STA  * $67        Ergebnis der Subtr. zurückschreiben
B96D:  A5 68     LDA  * $68        Akku mit OP3 Highest laden
B96F:  E9 00     SBC  # $00        Unterlauf der Middle Subtr. beachten
B971:  85 68     STA  * $68        Ergebnis der Subtr. zurückschreiben
B973:  60        RTS               Rücksprung aus dem Unterprogramm

*******************************    Kopiere bei gelöschtem Carry den
                                   Inhalt des OP1 in die Zero-Page
                                   Speicher für Bank-Nr, PC-Hi, PC-Lo

B974:  B0 0C     BCS  $B982        Bei gesetztem Carry Rücksprung
B976:  A5 60     LDA  * $60        Akku mit OP1 Lowest (Adr-Lo) laden
B978:  A4 61     LDY  * $61        Y-Reg mit OP1 Middle (Adr-Hi) laden
B97A:  A6 62     LDX  * $62        X-REG m. OP1 Highest (Bank-Byte) l.
B97C:  85 04     STA  * $04        In Zero-Page Byte für PC-Lo bringen
B97E:  84 03     STY  * $03        In Zero-Page Byte für PC-Hi bringen
B980:  86 02     STX  * $02        In Zero-Page Byte f. Bank-Nr bringen
B982:  60        RTS               Rücksprung aus dem Unterprogramm

*******************************    'Von' Operanden in OP3 bringen
                                   'Bis' Operanden in OP1 holen
                                   'Bis' Operanden in OPH kopieren
                                   Differenz aus OP1-OP3 bilden und als
                                   'Stepzahl' in OP1 ablegen
                                   'Stepzahl' in OP2 kopieren

B983:  B0 2A     BCS  $B9AF        Bei Fehler im Befehlsparameter: Exit
B985:  20 01 B9  JSR  $B901        Kopieren des OP1-Inhalts nach OP3
```

```
B988:  20 A7 B7   JSR   $B7A7      'Bis' Operanden in OP1 holen
B98B:  B0 22      BCS   $B9AF      'Bis' Operand ungültig, Fehler Exit
B98D:  A5 60      LDA   * $60      Kopiere den Inhalt
B98F:  8D B7 0A   STA   $0AB7      des 3 Byte-Operanden
B992:  A5 61      LDA   * $61      OP1 in den 3 Byte-
B994:  8D B8 0A   STA   $0AB8      Hilfsoperanden in
B997:  A5 62      LDA   * $62      den Speicherstellen
B999:  8D B9 0A   STA   $0AB9      $0AB9-$0AB8-$0AB7
B99C:  20 0E B9   JSR   $B90E      Differenz: OP1-OP3 in OP1 speichern
B99F:  A5 60      LDA   * $60      Kopiere den
B9A1:  85 63      STA   * $63      Inhalt des
B9A3:  A5 61      LDA   * $61      3 Byte OP1-
B9A5:  85 64      STA   * $64      Operanden
B9A7:  A5 62      LDA   * $62      in den OP2
B9A9:  85 65      STA   * $65      Operanden
B9AB:  90 02      BCC   $B9AF      Wenn OP1 größer OP3,dann Fehler Exit
B9AD:  18         CLC              Carry löschen als Kennzeichen für OK
B9AE:  24         .Byte  $24       Skip nach $B9B0 (RTS)

******************************     Routine Exit bei Fehler aufgetreten

B9AF:  38         SEC              Carry setzen = KZ f. Fehler gefunden
B9B0:  60         RTS              Rücksprung aus dem Unterprogramm

******************************     Ausgabe bei Umrechnungsbefehl (&%+$)

B9B1:  20 A5 B7   JSR   $B7A5      Den Umrechnungswert in OP1 holen
B9B4:  20 B9 B8   JSR   $B8B9      <Cr> <Esc-Q> <Blank> ausgeben
B9B7:  A9 24      LDA   # $24      Akku mit <$> Zeichen laden
B9B9:  20 D2 FF   JSR   $FFD2      Kernal BSOUT: Ein Zeichen ausgeben
B9BC:  A5 62      LDA   * $62      Lade Hi des 3 Byte-Umrechnungswertes
B9BE:  F0 07      BEQ   $B9C7      Bei $00,führende Nullen unterdrücken
B9C0:  20 D2 B8   JSR   $B8D2      Akku in 2 Byte ASCII Code: Hi=A,Lo=X
B9C3:  8A         TXA              ASCII Code für Lo Tetrade in Akku
B9C4:  20 D2 FF   JSR   $FFD2      Kernal BSOUT: Ein Zeichen ausgeben
B9C7:  A5 60      LDA   * $60      Lade Lo des 3 Byte-Umrechnungswertes
B9C9:  A6 61      LDX   * $61      Lade Mi des 3 Byte-Umrechnungswertes
B9CB:  20 9F B8   JSR   $B89F      Diese als 4 ASCII Zeichen ausgeben
B9CE:  20 B9 B8   JSR   $B8B9      <Cr> <Esc-Q> <Blank> ausgeben
B9D1:  A9 2B      LDA   # $2B      Akku mit <+> Zeichen laden
B9D3:  20 D2 FF   JSR   $FFD2      Kernal BSOUT: Ein Zeichen ausgeben
B9D6:  20 07 BA   JSR   $BA07      Konvertiere OP1 in Dezimalzahl
B9D9:  A9 00      LDA   # $00      KZ für führende Nullen unterdrücken
B9DB:  A2 08      LDX   # $08      8 Ziffern sind auszugeben
B9DD:  A0 03      LDY   # $03      Je 4 Bits bilden eine Ausgabeziffer
B9DF:  20 5D BA   JSR   $BA5D      AA0-AA3 als Dezimalzahl ausgeben
B9E2:  20 B9 B8   JSR   $B8B9      <Cr> <Esc-Q) <Blank> ausgeben
B9E5:  A9 26      LDA   # $26      Akku mit <&> Zeichen laden
```

```
B9E7:  20 D2 FF   JSR  $FFD2      Kernal BSOUT: Ein Zeichen ausgeben
B9EA:  A9 00      LDA  # $00      KZ für führende Nullen unterdrücken
B9EC:  A2 08      LDX  # $08      8 Ziffern sind auszugeben
B9EE:  A0 02      LDY  # $02      Je 3 Bits bilden eine Ausgabeziffer
B9F0:  20 47 BA   JSR  $BA47      AA0-AA3 als Oktalzahl ausgeben
B9F3:  20 B9 B8   JSR  $B8B9      <Cr> <Esc-q> <Blank> ausgeben
B9F6:  A9 25      LDA  # $25      Akku mit <%> Zeichen laden
B9F8:  20 D2 FF   JSR  $FFD2      Kernal BSOUT: Ein Zeichen ausgeben
B9FB:  A9 00      LDA  # $00      KZ für führende Nullen unterdrücken
B9FD:  A2 18      LDX  # $18      18 Ziffern sind auszugeben
B9FF:  A0 00      LDY  # $00      Je 1 Bit bildet eine Ausgabeziffer
BA01:  20 47 BA   JSR  $BA47      AA0-AA3 als Dualzahl ausgeben
BA04:  4C 8B B0   JMP  $B08B      Springe in Eingabe-Warteschleife

******************************    Den OP1-Inhalt in eine 8stellige
                                  Dezimalzahl in AA0-AA4 umwandeln

BA07:  20 01 B9   JSR  $B901      Kopieren des OP1 Inhalts nach OP3
BA0A:  A9 00      LDA  # $00      Den Bereich von AA0-AA3 für die
BA0C:  A2 07      LDX  # $07      aufbereitete Dezimalzahl löschen
BA0E:  9D A0 0A   STA  $0AA0,X    Den Bereich AA4-AA7 als Hilfszähler
BA11:  CA         DEX             für die Dezimalwandlung löschen.
BA12:  10 FA      BPL  $BA0E      Die Einerstelle des Hilfszählers
BA14:  EE A7 0A   INC  $0AA7      mit <1> initialisieren
BA17:  A0 17      LDY  # $17      Schleifenzähler für Umwandlungssteps
BA19:  08         PHP             Dezimal + Interrupt Status sichern
BA1A:  78         SEI             Alle System-Interrupts verhindern
BA1B:  F8         SED             Dezimale Betriebsart einschalten
BA1C:  46 68      LSR  * $68      Den 3 Byte-Ausgangswert
BA1E:  66 67      ROR  # $67      im Operanden (OP3)
BA20:  66 66      ROR  # $66      durch <2> dividieren
BA22:  90 0F      BCC  $BA33      Kein Rest, dann Skip Dezimaladdition
BA24:  18         CLC             Carry für Dezimaladdition löschen
BA25:  A2 03      LDX  # $03      Wenn bei der Division ein Rest
BA27:  BD A4 0A   LDA  $0AA4,X    aufgetreten ist, dann addiere den
BA2A:  7D A0 0A   ADC  $0AA0,X    Inhalt des 4-Byte-Hilfszählers,
BA2D:  9D A0 0A   STA  $0AA0,X    dessen Inhalt jeweils 2er Potenzen
BA30:  CA         DEX             sind, zum Inhalt des 4-Byte-Ausgabe-
BA31:  10 F4      BPL  $BA27      speichers. (4 Byte = 8 Ziffern)
BA33:  18         CLC             Carry für Dezimaladdition löschen
BA34:  A2 03      LDX  # $03      Multipliziere den Inhalt
BA36:  BD A4 0A   LDA  $0AA4,X    des 4 Byte Hilfszählers mit dem
BA39:  7D A4 0A   ADC  $0AA4,X    Faktor <2>. Somit ist der Inhalt
BA3C:  9D A4 0A   STA  $0AA4,X    des Hilfszählers immer die entsp.
BA3F:  CA         DEX             2er Potenz des Bits, das im OP3
BA40:  10 F4      BPL  $BA36      Operanden gerade bearbeitet wird.
BA42:  88         DEY             Schleifenzähler um 1 vermindern,
BA43:  10 D7      BPL  $BA1C      bis alle Steps abgearbeitet sind
```

```
BA45:  28        PLP             entspricht einem SED und CLI Befehl
BA46:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   Den 3-Byte-OP1-Operanden in den
                                 4-Byte-Ausgabeoperanden OPA wandeln

BA47:  48        PHA             Akku-Inhalt auf Stack sichern
BA48:  A5 60     LDA  * $60      Inhalt des OP1 Lo Bytes
BA4A:  8D A2 0A  STA  $0AA2      in OPA Middle-Lo kopieren
BA4D:  A5 61     LDA  * $61      Inhalt des OP1 Mi Bytes
BA4F:  8D A1 0A  STA  $0AA1      in OPA Middle-Hi kopieren
BA52:  A5 62     LDA  * $62      Inhalt des OP1 Hi Bytes
BA54:  8D A0 0A  STA  $0AA0      in OPA High kopieren
BA57:  A9 00     LDA  # $00      Akku mit $00 initialisieren und
BA59:  8D A3 0A  STA  $0AA3      in OPA Low kopieren
BA5C:  68        PLA             Akku Inhalt vom Stack zurückholen

******************************   Ausgabe des OPA-Operanden entsp.
                                 der Vorgabe im X-Reg und Y-Reg

BA5D:  8D B4 0A  STA  $0AB4      Flag für Nullunterdrückung setzen
BA60:  8C B6 0A  STY  $0AB6      Bitzahl für 1 Ausgabeziffer sichern
BA63:  AC B6 0A  LDY  $0AB6      Bitzahl für 1 Ausgabeziffer holen
BA66:  A9 00     LDA  # $00      Akku als Ausgabespeicher initial.
BA68:  0E A3 0A  ASL  $0AA3      Den Inhalt des 4-Byte
BA6B:  2E A2 0A  ROL  $0AA2      Ausgabeoperanden um eine
BA6E:  2E A1 0A  ROL  $0AA1      Bitposition nach links
BA71:  2E A0 0A  ROL  $0AA0      verschieben. Das herausgeschobene
BA74:  2A        ROL  A          höchstwertigste Bit im Akku sichern
BA75:  88        DEY             Bitzähler für 1 Ausgabeziffer - 1
BA76:  10 F0     BPL  $BA68      Schleifen bis eine Ziffer im Akku
BA78:  A8        TAY             Wert der Ausgabeziffer in Y sichern
BA79:  D0 09     BNE  $BA84      Wert ungleich 0, dann ausgeben
BA7B:  E0 01     CPX  # $01      Prüfe, ob es die 1er Stelle ist
BA7D:  F0 05     BEQ  $BA84      Ja, Ziffer auf jeden Fall ausgeben
BA7F:  AC B4 0A  LDY  $0AB4      Flag für Nullunterdrückung laden
BA82:  F0 08     BEQ  $BA8C      Noch aktiv, dann 0 nicht ausgeben
BA84:  EE B4 0A  INC  $0AB4      Nullunterdrückung abschalten
BA87:  09 30     ORA  # $30      ASCII Wert d. Ausgabeziffer erzeugen
BA89:  20 D2 FF  JSR  $FFD2      Kernal BSOUT: Ein Zeichen ausgeben
BA8C:  CA        DEX             Schleifenzähler für Anz. d. Ziffern
BA8D:  D0 D4     BNE  $BA63      Nicht 0, dann nächste Ziffer ausgeb.
BA8F:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   Monitor Befehl: @  (Disc Command)

BA90:  D0 03     BNE  $BA95      KZ für Geräteadresse vorhanden
BA92:  A2 08     LDX  # $08      Setze Standardgeräteadresse Floppy 8
```

```
BA94:  2C          .Byte  $2C      Skip nach $BA97

*****************************     Disc-Command-Routine mit Parameter
                                   für Geräteadresse

BA95:  A6 60       LDX  * $60      Geräteadresse aus OP1 Adr-Lo holen
BA97:  E0 04       CPX  # $04      Geräteadresse kleiner 4 ist ungültig
BA99:  90 65       BCC  $BB00      <?> ausgeben und in E-Warteschleife
BA9B:  E0 1F       CPX  # $1F      Geräteadresse größer 30 ist ungültig
BA9D:  B0 61       BCS  $BB00      <?> ausgeben und in E-Warteschleife
BA9F:  86 60       STX  * $60      Geräteadresse in OP1-Adr-Lo sichern
BAA1:  A9 00       LDA  # $00      Bank Nr. f. LSV und Dateinamen laden
BAA3:  85 62       STA  * $62      und in OP1-Bank-Byte sichern
BAA5:  85 B7       STA  * $B7      Länge des Dateinamens auf 0 setzen
BAA7:  AA          TAX             Akku+X-Reg f. SETBNK Routine löschen
BAA8:  20 68 FF    JSR  $FF68      Kernal SETBNK: Bank Nr.f.LSV+Datnam.
BAAB:  20 E9 B8    JSR  $B8E9      Lese ein Zeichen aus Eingabe-Buffer
BAAE:  C6 7A       DEC  * $7A      Displ. Zeiger E-Buf -1 (wie CHRGOT)
BAB0:  C9 24       CMP  # $24      Ist gelesenes Zeichen ein <$> ?
BAB2:  F0 4F       BEQ  $BB03      Ja, dann Directory ausgeben
BAB4:  A9 00       LDA  # $00      Logische Dateinummer (0) in Akku
BAB6:  A6 60       LDX  * $60      Geräteadresse aus OP1 Adr-Lo holen
BAB8:  A0 0F       LDY  # $0F      Sekundäradresse (15) setzen
BABA:  20 BA FF    JSR  $FFBA      Kernal SETLFS: Setze Dateiparameter
BABD:  20 C0 FF    JSR  $FFC0      Kernal OPEN: Datei öffnen
BAC0:  B0 32       BCS  $BAF4      Fehler beim OPEN, dann CLRCH u. Exit
BAC2:  A2 00       LDX  # $00      Logische Datei (0) a. Ausgabe setzen
BAC4:  20 C9 FF    JSR  $FFC9      Kernal CKOUT: Ausgabekanal setzen
BAC7:  B0 2B       BCS  $BAF4      Wenn Fehler aufgetreten, dann Exit
BAC9:  A6 7A       LDX  * $7A      Displ. Zeiger auf E-Buffer holen
BACB:  E6 7A       INC  * $7A      und auf nächstes Zeichen hochsetzen
BACD:  BD 00 02    LDA  $0200,X    Zeichen aus E-Buffer,Displ. lesen
BAD0:  F0 05       BEQ  $BAD7      Bei Bef-End Befehlskanal schließen
BAD2:  20 D2 FF    JSR  $FFD2      Kernal BSOUT: Ein Zeichen ausgeben
BAD5:  90 F2       BCC  $BAC9      OK, dann nächstes Zeichen ausgeben
BAD7:  20 CC FF    JSR  $FFCC      Kernal CLRCH: E/A Kanäle rücksetzen
BADA:  20 B4 B8    JSR  $B8B4      Zeilenvorschub + Clear Rest of Line
BADD:  A2 00       LDX  # $00      Logische Datei (0) a. Eingabe setzen
BADF:  20 C6 FF    JSR  $FFC6      Kernal CHKIN: Eingabekanal setzen
BAE2:  B0 10       BCS  $BAF4      Wenn Fehler aufgetreten, dann Exit
BAE4:  20 CF FF    JSR  $FFCF      Kernal BASIN: Ein Zeichen einlesen
BAE7:  20 D2 FF    JSR  $FFD2      Kernal BSOUT: Ein Zeichen ausgeben
BAEA:  C9 0D       CMP  # $0D      Wurde <Cr> ausgegeben ?
BAEC:  F0 06       BEQ  $BAF4      Ja, dann CLRCH und Exit Routine
BAEE:  A5 90       LDA  * $90      System STATUS in Akku laden
BAF0:  29 BF       AND  # $BF      Bit 6 (= Dateiende) ausmaskieren
BAF2:  F0 F0       BEQ  $BAE4      Kein Fehler, dann weiter einlesen
BAF4:  20 CC FF    JSR  $FFCC      Kernal CLRCH: E/A Kanäle rücksetzen
```

```
BAF7:  A9 00     LDA  # $00      Logische Datei (0) ganz schließen
BAF9:  38        SEC             Carry für CLOSE Routine setzen
BAFA:  20 C3 FF  JSR  $FFC3      Kernal CLOSE: Datei schließen
BAFD:  4C 8B B0  JMP  $B08B      Springe in Eingabe Warteschleife
BB00:  4C BC B0  JMP  $B0BC      <?> Ausgeben und zur E-Warteschleife

******************************   Routine für Disc Command Direktory

BB03:  A0 FF     LDY  # $FF      Längenzähler Dateiname a. -1 setzen
BB05:  A6 7A     LDX  * $7A      Displ. Zeiger auf E-Buffer holen
BB07:  CA        DEX             und auf vorheriges Zeichen setzen
BB08:  C8        INY             Längenzähler für Dateiname erhöhen
BB09:  E8        INX             Displ. Zeiger auf nächstes Zeichen
BB0A:  BD 00 02  LDA  $0200,X    Zeichen aus E-Buffer,Displ. lesen
BB0D:  D0 F9     BNE  $BB08      Kein Bef-End, dann nächstes Zeichen
BB0F:  98        TYA             Länge des Dateinamens in A kopieren
BB10:  A6 7A     LDX  * $7A      Adr-Lo des Dateinamens in X-Reg
BB12:  A0 02     LDY  # $02      Adr-Hi des Dateinamens in Y-Reg
BB14:  20 BD FF  JSR  $FFBD      Kernal SETNAM: Setze Dateinamen
BB17:  A9 00     LDA  # $00      Logische Dateinummer (0) in Akku
BB19:  A6 60     LDX  * $60      Geräteadresse aus OP1 Adr-Lo holen
BB1B:  A0 60     LDY  # $60      Sekundäradresse (96) setzen
BB1D:  20 BA FF  JSR  $FFBA      Kernal SETLFS: Setze Dateiparameter
BB20:  20 C0 FF  JSR  $FFC0      Kernal OPEN: Datei öffnen
BB23:  B0 CF     BCS  $BAF4      Wenn Fehler aufgetreten, dann Exit
BB25:  A2 00     LDX  # $00      Logische Datei (0) a. Eingabe setzen
BB27:  20 C6 FF  JSR  $FFC6      Kernal CHKIN: Eingabekanal setzen
BB2A:  20 B4 B8  JSR  $B8B4      Zeilenvorschub + Clear Rest of Line
BB2D:  A0 03     LDY  # $03      Zähler, um die ersten 6 Directory
BB2F:  84 63     STY  * $63      Bytes zu überlesen
BB31:  20 CF FF  JSR  $FFCF      Kernal BASIN: Ein Zeichen einlesen
BB34:  85 60     STA  * $60      Dir-Zeichen in OP1 Adr-Lo sichern
BB36:  A5 90     LDA  * $90      System STATUS in Akku laden
BB38:  D0 BA     BNE  $BAF4      Wenn Fehler aufgetreten, dann Exit
BB3A:  20 CF FF  JSR  $FFCF      Kernal BASIN: Ein Zeichen einlesen
BB3D:  85 61     STA  * $61      Dir-Zeichen in OP1 Adr-Hi sichern
BB3F:  A5 90     LDA  * $90      System STATUS in Akku laden
BB41:  D0 B1     BNE  $BAF4      Ist hier KZ für Dir-Ende,dann Exit
BB43:  C6 63     DEC  * $63      Dir-Bytes Skip Zähler runterzählen
BB45:  D0 EA     BNE  $BB31      Nicht 0, weitere Dir-Bytes überlesen
BB47:  20 07 BA  JSR  $BA07      Aufbereiten und ausgeben des OP1
BB4A:  A9 00     LDA  # $00      Inhalts in dezimaler Form. Gibt
BB4C:  A2 08     LDX  # $08      hier die Länge eines Directory
BB4E:  A0 03     LDY  # $03      Eintrages und die Anzahl der
BB50:  20 5D BA  JSR  $BA5D      freien Blöcke aus.
BB53:  A9 20     LDA  # $20      Akku mit <Blank> Zeichen laden
BB55:  20 D2 FF  JSR  $FFD2      Kernal BSOUT: Ein Zeichen ausgeben
BB58:  20 CF FF  JSR  $FFCF      Kernal BASIN: Ein Zeichen einlesen
```

```
BB5B:  F0 09         BEQ  $BB66          $0 ist KZ f. 1 Dir Zeile komplett
BB5D:  A6 90         LDX  * $90          System STATUS in X-Reg laden
BB5F:  D0 93         BNE  $BAF4          Wenn Fehler aufgetreten, dann Exit
BB61:  20 D2 FF      JSR  $FFD2          Kernal BSOUT: Ein Zeichen ausgeben
BB64:  90 F2         BCC  $BB58          Nächstes Zeichen der Dir Zeile ausg.
BB66:  20 B4 B8      JSR  $B8B4          Zeilenvorschub + Clear Rest of Line
BB69:  20 E1 FF      JSR  $FFE1          Kernal STOP: Auf Stop Taste prüfen
BB6C:  F0 86         BEQ  $BAF4          Wenn Stop gedrückt, dann Exit Rout.
BB6E:  A0 02         LDY  # $02          Zähler für 4 Dir Bytes überlesen
BB70:  D0 BD         BNE  $BB2F          Unbedingter Sprung zum Dir lesen

******************************           Ende des ROM-Monitor Bereichs

BB72:  FF FF FF . . .                    Füllwerte
BFFB:  . . . FF FF FF

BFFE:  00 3A

******************************           Sprungtabelle für Editor-Routinen

C000:  4C 7B C0      JMP  $C07B          CINT Initialisiert Editor + Screen
C003:  4C 34 CC      JMP  $CC34          DISPLAY Char in A, Farbe in X
C006:  4C 34 C2      JMP  $C234          LP2 Holt ein Zeichen vom IRQ-Buffer
C009:  4C 9B C2      JMP  $C29B          LOOP5 Ein Zeichen vom Bildschirm
C00C:  4C 2D C7      JMP  $C72D          PRINT Vektor für Bildschirmausgabe
C00F:  4C 5B CC      JMP  $CC5B          SCRORG Gibt Bildschirmbreite zurück
C012:  4C 87 FC      JMP  $FC87          KEY Tastaturabfrage
C015:  4C 51 C6      JMP  $C651          REPEAT Wiederholung der Tast.-Logik
C018:  4C 6A CC      JMP  $CC6A          PLOT Liest oder setzt Cursor-Pos.
C01B:  4C 57 CD      JMP  $CD57          CURSOR Bewegt 80 Zeichen Chip Cursor
C01E:  4C C1 C9      JMP  $C9C1          ESCAPE Ausgabe einer Esc-Sequenz
C021:  4C A2 CC      JMP  $CCA2          PFKEY Definiert eine Funktionstaste
C024:  4C 94 C1      JMP  $C194          IRQ Einsprung Editor IRQ-Routine
C027:  4C 0C CE      JMP  $CE0C          INIT80 Initialisiert 80Zeichen
C02A:  4C 2E CD      JMP  $CD2E          SWAPPER Tauscht 40/80 Zeichen
C02D:  4C 1B CA      JMP  $CA1B          WINDOW Setzt linke/obere bzw.
                                         rechte/untere Ecke des Fensters

C030:  FF FF FF                          Frei für künftige Erweiterungen

******************************           Zeilenanfänge Lo-Bytes

C033:  00 28 50 78 A0 C8 F0 18           $0400, $0428, $0450
C03B:  40 68 90 B8 E0 08 30 58           $0478, $04A0, $04C8
C043:  80 A8 D0 F8 20 48 70 98           $04F0, $0518, $0540
C04B:  C0                                $0568, $0590, $05B8
```

```
******************************   Zeilenanfänge Hi-Bytes

C04C:  04 04 04 04 04 04 04 05   $05E0, $0608, $0630
C054:  05 05 05 05 05 06 06 06   $0658, $0680, $06A8
C05C:  06 06 06 06 07 07 07 07   $06D0, $06F8, $0720
C064:  07                        $0748, $0770, $0798, $07C0

******************************   Zeichenausgabe- und Tastaturvektoren

C065:  B9 C7     ($C7B9)         Einsprung: Zeichenausgabe mit Ctrl
C067:  05 C8     ($C805)         Einsprung: Zeichenausgabe mit Shift
C069:  C1 C9     ($C9C1)         Einsprung: Zeichenausgabe mit Esc
C06B:  E1 C5     ($C5E1)         Einsprung: Tastatur auswerten
C06D:  AD C6     ($C6AD)         Einsprung: Tastendruck speichern

******************************   Zeiger auf Tastatur Decodiertabellen

C06F:  80 FA     ($FA80)         Tastatur Decodiertabelle 1a
C071:  D9 FA     ($FAD9)         Tastatur Decodiertabelle 2a
C073:  32 FB     ($FB32)         Tastatur Decodiertabelle 3a
C075:  8B FB     ($FB8B)         Tastatur Decodiertabelle 4a
C077:  80 FA     ($FA80)         Tastatur Decodiertabelle 1a
C079:  E4 FB     ($FBE4)         Tastatur Decodiertabelle 5a

******************************   Kernal Routine: CINT
                                 Initialisiere Editor und Bildschirm

C07B:  A9 03     LDA  # $03      Zwei höchstwertigen Bits v. Basis
C07D:  0D 00 DD  ORA  $DD00      Video-Ram setzen, da LO-aktiv
C080:  8D 00 DD  STA  $DD00      und wieder abspeichern
C083:  A9 FB     LDA  # $FB      Bit 2 des Datenrichtungsregisters
C085:  25 01     AND  * $01      löschen und danach Bit 1 des Daten-
C087:  09 02     ORA  # $02      richtungsregisters setzen und
C089:  85 01     STA  * $01      wieder abspeichern
C08B:  20 CC FF  JSR  $FFCC      Kernal CLRCH: E/A Kanäle rücksetzen
C08E:  A9 00     LDA  # $00      Filter, Lautstärke und Eintrag in
C090:  20 80 FC  JSR  $FC80      Tabelle f.Logged in Cards rücksetzen
C093:  85 D8     STA  * $D8      Textbildschirmflag auf "Text" setzen
C095:  85 D7     STA  * $D7      40/80 Zeichen Flag auf "40" setzen
C097:  85 D0     STA  * $D0      Tastaturpuffer Warteschlange löschen
C099:  85 D1     STA  * $D1      Funktionstastenflag löschen
C09B:  85 D6     STA  * $D6      Tastatur Input/Get Flag zurücksetzen
C09D:  8D 21 0A  STA  $0A21      Pause (Ctrl-S) Flag zurücksetzen
C0A0:  8D 26 0A  STA  $0A26      Flag Cursor in Blinkphase rücksetzen
C0A3:  85 D9     STA  * $D9      Zeiger für Zeichensatz im RAM/ROM
C0A5:  8D 2E 0A  STA  $0A2E      Basisadresse d. Bildschirm-Text-RAM
C0A8:  A9 14     LDA  # $14      Initialisierungswert f. Basis Zeiger
C0AA:  8D 2C 0A  STA  $0A2C      Text Bildschirm/Zeichen Basis Zeiger
```

```
C0AD:  A9 78     LDA  # $78     Initialisierungswert Bit Map Basis
C0AF:  8D 2D 0A  STA  $0A2D     Bit Map Basis Zeiger initialisieren
C0B2:  A9 08     LDA  # $08     Initialisierungswert Attribut-RAM
C0B4:  8D 2F 0A  STA  $0A2F     Attribut-RAM-Basis initialisieren
C0B7:  AD 4C C0  LDA  $C04C     Initialisierungswert ($04) laden
C0BA:  8D 3B 0A  STA  $0A3B     PAL-System-Zeiger initialisieren
C0BD:  A9 0A     LDA  # $0A     Startwert d. Tastaturpuffergröße
C0BF:  8D 20 0A  STA  $0A20     Flag für Tastaturpuffergröße init.
C0C2:  8D 28 0A  STA  $0A28     Zählzeiger für blinkenden Cursor
C0C5:  8D 27 0A  STA  $0A27     Flag für Cursor-Blinkmodus
C0C8:  8D 24 0A  STA  $0A24     Flag: Tastenwiederholverzögerung
C0CB:  A9 04     LDA  # $04     Startwert für Zählgeschwindigkeit
C0CD:  8D 23 0A  STA  $0A23     Flag: Zählgeschwindigkeit Wiederhlg.
C0D0:  20 83 C9  JSR  $C983     TAB Positionen initialisieren
C0D3:  8D 22 0A  STA  $0A22     Flag für Tastenwiederholungszeiger
C0D6:  0D 05 D5  ORA  $D505     Das Fast-Seriell-Controll-Bit in das
C0D9:  8D 05 D5  STA  $D505     MCR der MMU einblenden
C0DC:  A9 60     LDA  # $60     Startwert für aktuellen Cursor Modus
C0DE:  8D 2B 0A  STA  $0A2B     Flag für aktuellen Cursor Modus
C0E1:  A9 D0     LDA  # $D0     Initialisierungswert für den System-
C0E3:  8D 34 0A  STA  $0A34     zeiger: Zeile löschen/verschieben
C0E6:  A2 1A     LDX  # $1A     Schleifenzähler f. Z-Page init.
C0E8:  BD 74 CE  LDA  $CE74,X   ROM-Kopie der 40-Zeichen Bildschirm
C0EB:  95 E0     STA  * $E0,X   Startwerte in Z-Page kopieren
C0ED:  BD 8E CE  LDA  $CE8E,X   ROM-Kopie der 80-Zeichen Bildschirm
C0F0:  9D 40 0A  STA  $0A40,X   Startwerte ins RAM kopieren
C0F3:  CA        DEX            Schleifenzähler um 1 vermindern
C0F4:  10 F2     BPL  $C0E8     Schleifen, bis alle Werte übertragen
C0F6:  A2 09     LDX  # $09     Schleifenzähler für Page 3 init.
C0F8:  BD 65 C0  LDA  $C065,X   ROM Kopie der Zeichen- und Tastatur
C0FB:  9D 34 03  STA  $0334,X   Vektoren in den RAM Bereich kopieren
C0FE:  CA        DEX            Schleifenzähler um 1 vermindern
C0FF:  10 F7     BPL  $C0F8     Schleifen, bis alle Werte übertragen
C101:  2C 04 0A  BIT  $0A04     Prüfe Bit 6 d. Initialisierungsflags
C104:  70 1E     BVS  $C124     Bit gesetzt, dann Skip
C106:  A2 0B     LDX  # $0B     Schleifenzähler für Page 3 init.
C108:  BD 6F C0  LDA  $C06F,X   ROM Kopie der Tastatur Dekodiertab.
C10B:  9D 3E 03  STA  $033E,X   Vektoren in den RAM Bereich kopieren
C10E:  CA        DEX            Schleifenzähler um 1 vermindern
C10F:  10 F7     BPL  $C108     Schleifen, bis alle Werte übertragen
C111:  A2 4C     LDX  # $4C     Schleifenzähler für F-Tasten init.
C113:  BD A8 CE  LDA  $CEA8,X   ROM Kopie der F-Tasten Längen und
C116:  9D 00 10  STA  $1000,X   Strings in RAM-Bereich kopieren
C119:  CA        DEX            Schleifenzähler um 1 vermindern
C11A:  10 F7     BPL  $C113     Schleifen, bis alle Werte übertragen
C11C:  A9 40     LDA  # $40     Bit 6 auf "ON" setzen und mit dem
C11E:  0D 04 0A  ORA  $0A04     Initialisierungs Flag verknüpfen
C121:  8D 04 0A  STA  $0A04     Ergebnis in Init. Flag ablegen
```

```
C124:  20 2E CD   JSR  $CD2E      Umschalten des 40/80-Zeichen-Modus
C127:  20 83 C9   JSR  $C983      Rücksetzen der Tabulatoren
C12A:  20 24 CA   JSR  $CA24      Window=gesamter Bildschirm
C12D:  20 42 C1   JSR  $C142      CLR/HOME
C130:  20 2E CD   JSR  $CD2E      Umschalten des 40/80-Zeichen-Modus
C133:  20 24 CA   JSR  $CA24      Window=gesamter Bildschirm
C136:  20 42 C1   JSR  $C142      CLR/HOME
C139:  2C 05 D5   BIT  $D505      Teste, ob 40/80-Zeichen-Modus
C13C:  30 03      BMI  $C141      getestet und springe, wenn 80
C13E:  20 2E CD   JSR  $CD2E      Umschalten des 40/80-Zeichen-Modus
C141:  60         RTS             Rücksprung aus dem Unterprogramm

******************************    Fenster löschen (CLR/HOME)

C142:  20 50 C1   JSR  $C150      Cursor Home
C145:  20 5E C1   JSR  $C15E      Startadresse von Zeile X errechnen
C148:  20 A5 C4   JSR  $C4A5      Lösche Zeile X
C14B:  E4 E4      CPX  * $E4      Vergleiche mit unterer Fenstergrenze
C14D:  E8         INX             Erhöhe Zeilenzeiger
C14E:  90 F5      BCC  $C145      Wenn untere Grenze nicht erreicht

******************************    Cursor Home im Fenster

C150:  A6 E5      LDX  * $E5      Obere Fenstergrenze in X-Reg laden
C152:  86 EB      STX  * $EB      aktuelle Cursor Zeile zurückschreib.
C154:  86 E8      STX  * $E8      Als Starteingabezeile speichern
C156:  A4 E6      LDY  * $E6      Linke Fenstergrenze in Y-Reg laden
C158:  84 EC      STY  * $EC      Sichern der aktuellen Cursor Spalte
C15A:  84 E9      STY  * $E9      und als Starteingabespalte

******************************    Adresse aktuelle Zeile setzen

C15C:  A6 EB      LDX  * $EB      Hole aktuelle Cursor Zeile ins X-Reg
C15E:  BD 33 C0   LDA  $C033,X    Hole Lo-Byte von Startzeile
C161:  24 D7      BIT  * $D7      Teste 40/80-Zeichen-Modus
C163:  10 01      BPL  $C166      Springe, wenn 40-Zeichen-Modus
C165:  0A         ASL  A          sonst Adresse mal 2
C166:  85 E0      STA  * $E0      Lo-Byte speichern
C168:  BD 4C C0   LDA  $C04C,X    Hole Hi-Byte der Startzeile
C16B:  29 03      AND  # $03      Bits 2-7 ausmaskieren=X MOD 4
C16D:  24 D7      BIT  * $D7      Teste 40/80-Zeichen-Modus
C16F:  10 06      BPL  $C177      Springe, wenn 40-Zeichen-Modus
C171:  2A         ROL  A          sonst shifte Übertrag ins Hi-Byte
C172:  0D 2E 0A   ORA  $0A2E      und addiere zu Video-Startadresse
C175:  90 03      BCC  $C17A      unbedingter Sprung nach $C17A
C177:  0D 3B 0A   ORA  $0A3B      Video-Startadresse bei 40 Zeichen
C17A:  85 E1      STA  * $E1      Hi-Byte abspeichern
```

```
******************************       Attribut-RAM-Adresse anpassen

C17C:  A5 E0     LDA  * $E0       Aktuelle Bildschirmzeile, Lo-Byte
C17E:  85 E2     STA  * $E2       nach Lo-Byte von Attribut-Adresse
C180:  A5 E1     LDA  * $E1       Hole Hi-Byte von akt. Bildschirmzeile
C182:  24 D7     BIT  * $D7       Teste auf 40/80-Zeichen-Modus
C184:  10 07     BPL  $C18D       40-Zeichen-Modus ist aktiv
C186:  29 07     AND  # $07       Bits 3-7 ausmaskieren
C188:  0D 2F 0A  ORA  $0A2F       Addiere Attribut RAM-Basis
C18B:  D0 04     BNE  $C191       Springe unbedingt
C18D:  29 03     AND  # $03       Bits 2-7 ausmaskieren
C18F:  09 D8     ORA  # $D8       Addiere Basis des Farb-RAMs
C191:  85 E3     STA  * $E3       Abspeichern des Attribut-Hi-Bytes
C193:  60        RTS              Rücksprung aus dem Unterprogramm

******************************       IRQ-Routine

C194:  38        SEC              Setze Carry-Flag als FLAG
C195:  AD 19 D0  LDA  $D019       Lade IRR vom VIC
C198:  29 01     AND  # $01       Teste Rasterzeilen-Interrupt-Bit
C19A:  F0 07     BEQ  $C1A3       Wenn nicht gesetzt, dann Sprung
C19C:  8D 19 D0  STA  $D019       Löschen des Registers
C19F:  A5 D8     LDA  * $D8       Text/Grafik testen
C1A1:  C9 FF     CMP  # $FF       Wenn Grafik Bildschirm eingeschaltet
C1A3:  F0 6F     BEQ  $C214       dann in entsprechende Routine
C1A5:  2C 11 D0  BIT  $D011       Teste Steuerregister 1 des VIC
C1A8:  30 04     BMI  $C1AE       Hi-Byte Rasterzeile ist gesetzt
C1AA:  29 40     AND  # $40       Teste Extend-Color-Modus
C1AC:  D0 31     BNE  $C1DF       Ist gesetzt
C1AE:  38        SEC              Setze Carry als Flag
C1AF:  A5 D8     LDA  * $D8       Hole Text/Grafik-Modus
C1B1:  F0 2C     BEQ  $C1DF       Textmodus - Sprung
C1B3:  24 D8     BIT  * $D8       Teste Text/Grafik-Modus
C1B5:  50 06     BVC  $C1BD       Bit 6=0 bedeutet keine Rasterzeilen
C1B7:  AD 34 0A  LDA  $0A34       IRQ. Sonst hole Rasterzeile und
C1BA:  8D 12 D0  STA  $D012       speichere erneut ab
C1BD:  A5 01     LDA  * $01       Hole Datenrichtungsregister und
C1BF:  29 FD     AND  # $FD       maskiere Bits 0-1 aus
C1C1:  09 04     ORA  # $04       Setze Bit 2 des Registers
C1C3:  48        PHA              und rette Konfiguration auf Stack
C1C4:  AD 2D 0A  LDA  $0A2D       Basisadresse der Grafik
C1C7:  48        PHA              Rette Basisadresse auf Stack
C1C8:  AD 11 D0  LDA  $D011       Hole Steuerregister 1 des VIC
C1CB:  29 7F     AND  # $7F       Übertrag Rasterzeile löschen und
C1CD:  09 20     ORA  # $20       Standard-Bitmap-Mode setzen
C1CF:  A8        TAY              Steuerregister nach Y
C1D0:  AD 16 D0  LDA  $D016       Hole Steuerregister 2 des VIC
C1D3:  24 D8     BIT  * $D8       Teste Text/Grafik-Register
```

```
C1D5:  30 03     BMI  $C1DA      Multi-Color-Modus gesetzt
C1D7:  29 EF     AND  # $EF      Lösche Multi-Color-Bit
C1D9:  2C        .Byte  $2C      Skip nach $C1DC
C1DA:  09 10     ORA  # $10      Setze Multi-Color-Bit
C1DC:  AA        TAX             Steuerregister 2 nach X
C1DD:  D0 28     BNE  $C207      Springe unbedingt

******************************   Textmodus

C1DF:  A9 FF     LDA  # $FF      Rasterzeile ist letzte Zeile
C1E1:  8D 12 D0  STA  $D012      Als Rasterzeile speichern
C1E4:  A5 01     LDA  * $01      Datenrichtungsregister holen
C1E6:  09 02     ORA  # $02      Bit 1 des Registers setzen
C1E8:  29 FB     AND  # $FB      und Bit 2 löschen
C1EA:  05 D9     ORA  * $D9      Bit 2 wird evtl. wieder gesetzt,
C1EC:  48        PHA             wenn CHARROM im RAM. Auf Stack
C1ED:  AD 2C 0A  LDA  $0A2C      Basisadresse von Text/Grafik
C1F0:  48        PHA             auch auf Stack sichern
C1F1:  AD 11 D0  LDA  $D011      Hole Steuerregister des VIC
C1F4:  29 5F     AND  # $5F      Übertrag und Grafik löschen
C1F6:  A8        TAY             Steuerregister 1 nach Y
C1F7:  AD 16 D0  LDA  $D016      Hole Steuerregister 2 des VIC
C1FA:  29 EF     AND  # $EF      Lösche Multi-Color-Bit
C1FC:  AA        TAX             Steuerregister 2 nach X
C1FD:  B0 08     BCS  $C207      Carry gesetzt=nicht warten
C1FF:  A2 07     LDX  # $07      X ist Zähler für Warteschleife
C201:  CA        DEX             Erniedrige den Zähler
C202:  D0 FD     BNE  $C201      und springe, wenn nicht Ende
C204:  EA        NOP             Noch zwei NOPs, um die Warteschleife
C205:  EA        NOP             zu perfektionieren
C206:  AA        TAX             Steuerregister 2 wieder nach X

******************************   Setzen der IRQ-Register

C207:  68        PLA             Hole Basisadresse zurück
C208:  8D 18 D0  STA  $D018      und Basisadresse nach VIC
C20B:  68        PLA             Hole Datenrichtungsregister von
C20C:  85 01     STA  * $01      Stack und abspeichern
C20E:  8C 11 D0  STY  $D011      Steuerregister 1 nach VIC
C211:  8E 16 D0  STX  $D016      und Steuerregister 2 nach VIC
C214:  B0 13     BCS  $C229      Wenn Carry gesetzt, dann Skip
C216:  AD 30 D0  LDA  $D030      Hole 1/2 Mhz Taktregister
C219:  29 01     AND  # $01      maskiere relevantes Bit aus
C21B:  F0 0C     BEQ  $C229      Springe, wenn 1 Mhz
C21D:  A5 D8     LDA  * $D8      Hole Text/Grafik-Modus
C21F:  29 40     AND  # $40      Teste Rasterzeilen-Interrupt-Bit
C221:  F0 06     BEQ  $C229      Kein Rasterzeilen-Interrupt
C223:  AD 11 D0  LDA  $D011      Hole Steuerregister 1
```

```
C226:  10 01     BPL  $C229     Kein Übertrag - Sprung
C228:  38        SEC            Setze Carry als Flag
C229:  58        CLI            Alle System Interrupts freigeben
C22A:  90 07     BCC  $C233     Ende, wenn Flag nicht gesetzt
C22C:  20 87 FC  JSR  $FC87     Aufruf der Kernal-Routine KEY
C22F:  20 E7 C6  JSR  $C6E7     VIC-Cursor blinken lassen
C232:  38        SEC            Carry setzen für OK
C233:  60        RTS            Rücksprung aus dem Unterprogramm

******************************  Zeichen aus KEY holen

C234:  A6 D1     LDX  * $D1     Müssen Zeichen aus KEY-Buffer
C236:  F0 0C     BEQ  $C244     geholt werden? NEIN
C238:  A4 D2     LDY  * $D2     Hole Pointer aus KEY-Buffer
C23A:  B9 0A 10  LDA  $100A,Y   Hole Zeichen aus KEY-Tabelle
C23D:  C6 D1     DEC  * $D1     erniedrige den Zeichenzähler
C23F:  E6 D2     INC  * $D2     erhöhe den Pointer
C241:  58        CLI            Alle System Interrupts freigeben
C242:  18        CLC            Lösche Carry für "Zeichen geholt"
C243:  60        RTS            Rücksprung aus dem Unterprogramm

******************************  Zeichen aus Buffer holen

C244:  AC 4A 03  LDY  $034A     Wieviele Zeichen sind im Queue?
C247:  BD 4B 03  LDA  $034B,X   Hole Zeichen aus Queue
C24A:  9D 4A 03  STA  $034A,X   und nach vorne schieben
C24D:  E8        INX            erhöhe den Zähler und verschiebe
C24E:  E4 D0     CPX  * $D0     Zeichen, bis alle Zeichen im
C250:  D0 F5     BNE  $C247     Queue nach vorne verschoben sind
C252:  C6 D0     DEC  * $D0     Offset der KEY-BUFFER Schlange -1
C254:  98        TYA            Zeichen nach Akku
C255:  58        CLI            Alle System Interrupts freigeben
C256:  18        CLC            Lösche Carry für "Zeichen geholt"
C257:  60        RTS            Rücksprung aus dem Unterprogramm

******************************  Eingabezeile holen (mit <CR>) LOOP4

C258:  20 2D C7  JSR  $C72D     Zeichen ausgeben
C25B:  20 6F CD  JSR  $CD6F     Cursor bewegen
C25E:  A5 D0     LDA  * $D0     Anzahl Zeichen im Tastaturbuffer
C260:  05 D1     ORA  * $D1     plus Anzahl Zeichen im KEY-Buffer
C262:  F0 FA     BEQ  $C25E     Wenn leer, dann warte
C264:  20 9F CD  JSR  $CD9F     Cursor setzen
C267:  20 34 C2  JSR  $C234     Zeichen aus Buffer holen
C26A:  C9 0D     CMP  # $0D     ist Zeichen <CR>
C26C:  D0 EA     BNE  $C258     Nein, dann hole nächstes Zeichen
C26E:  85 D6     STA  * $D6     Input-Flag setzen
C270:  A9 00     LDA  # $00     Cursor Mode Flag
```

```
C272:  85 F4     STA  * $F4    löschen
C274:  20 C3 CB  JSR  $CBC3    Ende Eingabezeile ermitteln
C277:  8E 30 0A  STX  $0A30    letzte Spaltenpos. merken
C27A:  20 B5 CB  JSR  $CBB5    Zeilenanfang setzen
C27D:  A4 E6     LDY  * $E6    Linke Fenstergrenze in Y-Reg laden
C27F:  A5 E8     LDA  * $E8    Start der laufenden Eingabezeile
C281:  30 13     BMI  $C296    lfd. Eingabezeile ist Folgezeile
C283:  C5 EB     CMP  * $EB    vergleiche m. aktueller Cursor Zeile
C285:  90 0F     BCC  $C296    Grenze noch nicht erreicht
C287:  A4 E9     LDY  * $E9    Start der laufenden Eingabespalte
C289:  CD 30 0A  CMP  $0A30    Vergleich mit letzter Eingabespalte
C28C:  D0 04     BNE  $C292    Ist nicht die gleiche Spalte
C28E:  C4 EA     CPY  * $EA    Mit Ende lfd. E-Zeile vergleichen
C290:  F0 02     BEQ  $C294    Ist erreicht
C292:  B0 11     BCS  $C2A5    Input/Get-Flag auf Get setzen
C294:  85 EB     STA  * $EB    aktuelle Cursor Zeile zurückschreib.
C296:  84 EC     STY  * $EC    sichern der aktuellen Cursor Spalte
C298:  4C BC C2  JMP  $C2BC    Zeichen an Cursor-Pos. holen

*******************************    Ein Zeichen vom Bildschirm holen

C29B:  98        TYA           Y-Register (Spalte) über Akku
C29C:  48        PHA           auf Stack sichern
C29D:  8A        TXA           X-Register (Zeile) über
C29E:  48        PHA           Akku auf Stack sichern
C29F:  A5 D6     LDA  * $D6    Input/Get-Flag holen
C2A1:  F0 B8     BEQ  $C25B    Bei GET zur Warteschleife
C2A3:  10 17     BPL  $C2BC    Noch kein <CR> erforderlich
C2A5:  A9 00     LDA  # $00    Es wird das Input/Get-Flag
C2A7:  85 D6     STA  * $D6    über Akku zurückgesetzt
C2A9:  A9 0D     LDA  # $0D    ASCII-Code für <CR>
C2AB:  A2 03     LDX  # $03    Code für Bildschirm mit
C2AD:  E4 99     CPX  * $99    Standardeingabegerät vergleichen
C2AF:  F0 04     BEQ  $C2B5    Eingabegerät ist Bildschirm
C2B1:  E4 9A     CPX  * $9A    mit St.-Ausgabegerät vergleichen
C2B3:  F0 03     BEQ  $C2B8    Ausgabe auf Bildschirm
C2B5:  20 2D C7  JSR  $C72D    BSOUT-Einsprung Bildschirm
C2B8:  A9 0D     LDA  # $0D    ASCII-Code für <CR>
C2BA:  D0 39     BNE  $C2F5    unbedingter Sprung zum Ende

*******************************    Zeichen an Cursorpos in ASCII

C2BC:  20 5C C1  JSR  $C15C    Adresse aktuelle Zeile holen
C2BF:  20 58 CB  JSR  $CB58    Zeichen und Farbe an Cursorpos.
C2C2:  85 EF     STA  * $EF    Zwischenspeicher für Druckzeichen
C2C4:  29 3F     AND  # $3F    Bits 6/7 ausmaskieren
C2C6:  06 EF     ASL  * $EF    Das Zeichen wird nun nach
C2C8:  24 EF     BIT  * $EF    ASCII gewandelt
```

```
C2CA:  10 02     BPL  $C2CE      kein reverses Zeichen
C2CC:  09 80     ORA  # $80      Setzen von Bit 7
C2CE:  90 04     BCC  $C2D4      Teste ehemaliges Bit 7
C2D0:  A6 F4     LDX  * $F4      Flag für Quote-Mode aktiv
C2D2:  D0 04     BNE  $C2D8      Ist aktiv, dann Sprung
C2D4:  70 02     BVS  $C2D8      Teste ehemaliges Bit 6
C2D6:  09 40     ORA  # $40      Setze Bit 6 für ASCII
C2D8:  20 FF C2  JSR  $C2FF      Teste auf (") und setzte Flags
C2DB:  A4 EB     LDY  * $EB      Hole aktuelle Cursor Zeile in Y-Reg
C2DD:  CC 30 0A  CPY  $0A30      letzte Spalte bereits erreicht?
C2E0:  90 0A     BCC  $C2EC      Nein, noch nicht
C2E2:  A4 EC     LDY  * $EC      Hole aktuelle Cursor Spalte in Y-Reg
C2E4:  C4 EA     CPY  * $EA      Vergleiche mit Ende
C2E6:  90 04     BCC  $C2EC      Endezeile noch nicht erreicht
C2E8:  66 D6     ROR  # $D6      Schiebe Carry ins 7. Bit von $D6
C2EA:  30 03     BMI  $C2EF      Wenn gesetzt, dann neue Zeile
C2EC:  20 ED CB  JSR  $CBED      Cursor um eine Position nach rechts
C2EF:  C9 DE     CMP  # $DE      Vergleiche auf ASCII "PI"
C2F1:  D0 02     BNE  $C2F5      Ist nicht Pi
C2F3:  A9 FF     LDA  # $FF      Sonst lade angepaßten Code PI
C2F5:  85 EF     STA  * $EF      als Druckzeichen speichern
C2F7:  68        PLA             Hole X-Register (Zeile) über
C2F8:  AA        TAX             Akku von Stack
C2F9:  68        PLA             Hole Y-Register (Spalte)
C2FA:  A8        TAY             über Akku von Stack
C2FB:  A5 EF     LDA  * $EF      Druckzeichen aus Zwischenspeicher
C2FD:  18        CLC             Flag für OK
C2FE:  60        RTS             Rücksprung aus dem Unterprogramm

*****************************    Teste auf (") und setze Flags

C2FF:  C9 22     CMP  # $22      Vergleich auf Anführungszeichen
C301:  D0 08     BNE  $C30B      Anderes Zeichen, dann Ende
C303:  A5 F4     LDA  * $F4      Hole aktuellen Quote-Modus
C305:  49 01     EOR  # $01      Kehre Modus um
C307:  85 F4     STA  * $F4      Und speicher wieder ab
C309:  A9 22     LDA  # $22      Lade Akku wieder mit ASCII-Wert
C30B:  60        RTS             Rücksprung aus dem Unterprogramm

*****************************    BSOUT Fortsetzung

C30C:  A5 EF     LDA  * $EF      Aktuell zu druckendes Zeichen als
C30E:  85 F0     STA  * $F0      zuletzt gedrucktes Zeichen merken
C310:  20 57 CD  JSR  $CD57      Setze Cursor auf aktuelle Spalte
C313:  A5 F5     LDA  * $F5      Hole Insert-Mode-Flag
C315:  F0 02     BEQ  $C319      Insert-Modus ist nicht aktiv
C317:  46 F4     LSR  * $F4      Shifte Quote-Modus-Flag
C319:  68        PLA             Hole ersten Wert von Stack
```

```
C31A:  A8          TAY              und nach Y-Register
C31B:  68          PLA              Hole zweiten Wert von Stack
C31C:  AA          TAX              Und nach X-Register
C31D:  68          PLA              Hole noch Akku von Stack
C31E:  18          CLC              Lösche Carry für OK
C31F:  60          RTS              Rücksprung aus dem Unterprogramm

******************************      Umrechnung von ASCII in POKE-Code

C320:  09 40       ORA  # $40       Setzte Bit 2 des Akkus
C322:  A6 F3       LDX  * $F3       Hole Flag für RVS Modus aktiv on/off
C324:  F0 02       BEQ  $C328       Nicht reverses Zeichen
C326:  09 80       ORA  # $80       Setze höchstwertiges Bit (Revers)
C328:  A6 F5       LDX  * $F5       Insert-Modus-Flag
C32A:  F0 02       BEQ  $C32E       Kein Insert-Modus
C32C:  C6 F5       DEC  * $F5       Erniedrige den Zähler
C32E:  24 F6       BIT  * $F6       Teste Auto-Insert-Flag
C330:  10 09       BPL  $C33B       Springe, wenn nicht aktiv
C332:  48          PHA              Rette Akku auf Stack
C333:  20 E3 C8    JSR  $C8E3       Verschiebe Bildschirm hinter Cursor
C336:  A2 00       LDX  # $00       Setze Insert-Mode-Flag
C338:  86 F5       STX  * $F5       auf Null zurück
C33A:  68          PLA              Hole Akku wieder von Stack
C33B:  20 2F CC    JSR  $CC2F       Zeichen an aktueller Pos. ausgeben

******************************      Cursor ans Zeilenende

C33E:  C4 E7       CPY  * $E7       Vergleiche mit rechter Fenstergrenze
C340:  90 0A       BCC  $C34C       Rechter Rand noch nicht erreicht
C342:  A6 EB       LDX  * $EB       Hole aktuelle Cursor Zeile in X-Reg
C344:  E4 E4       CPX  * $E4       Vergleiche mit unterer Fenstergrenze
C346:  90 04       BCC  $C34C       Untere Grenze noch nicht erreicht
C348:  24 F8       BIT  * $F8       Teste Scroll-Flag
C34A:  30 16       BMI  $C362       Kein Scrolling, dann Ende
C34C:  20 5C C1    JSR  $C15C       Ermittle Startadresse akt. Zeile
C34F:  20 ED CB    JSR  $CBED       Cursor ein Zeichen nach rechts
C352:  90 0E       BCC  $C362       Keine neue Zeile
C354:  20 74 CB    JSR  $CB74       Zeilenüberlaufbit testen
C357:  B0 08       BCS  $C361       Zeilenüberlaufbit ist gesetzt
C359:  38          SEC              Setze Carry für kein Scrolling
C35A:  24 F8       BIT  * $F8       Teste Scroll-Bit
C35C:  70 04       BVS  $C362       Springe, wenn kein Scrolling
C35E:  20 7C C3    JSR  $C37C       Zeile an X einfügen
C361:  18          CLC              Lösche Carry für gescrollt
C362:  60          RTS              Rücksprung aus dem Unterprogramm
```

```
******************************     Zeilenvorschub ausführen

C363:  A6 EB     LDX  * $EB      Hole aktuelle Cursor Zeile in X-Reg
C365:  E4 E4     CPX  * $E4      Vergleiche mit unterer Fenstergrenze
C367:  90 0E     BCC  $C377      Untere Grenze noch nicht erreicht
C369:  24 F8     BIT  * $F8      Teste Scroll-Bit
C36B:  10 06     BPL  $C373      Scrollen möglich
C36D:  A5 E5     LDA  * $E5      Obere Fenstergrenze in Akku laden
C36F:  85 EB     STA  * $EB      aktuelle Cursor Zeile zurückschreib.
C371:  B0 06     BCS  $C379      Unbedingter Sprung nach $C379
C373:  20 A6 C3  JSR  $C3A6      Scrolling
C376:  18        CLC             Carry löschen für Ok, gescrollt
C377:  E6 EB     INC  * $EB      aktuelle Cursor Zeile um 1 erhöhen
C379:  4C 5C C1  JMP  $C15C      Ermittle Startadresse akt. Zeile

******************************     Zeile einfügen (an Zeile X)

C37C:  A6 E8     LDX  * $E8      Start der lfd. Eingabezeile
C37E:  30 06     BMI  $C386      Zeile ist eine Folgezeile
C380:  E4 EB     CPX  * $EB      vergleiche m. aktueller Cursor Zeile
C382:  90 02     BCC  $C386      Cursor-Zeile erreicht?
C384:  E6 E8     INC  * $E8      Erhöhe den Start der lfd. E-Zeile
C386:  A6 E4     LDX  * $E4      Untere Fenstergrenze in X-Reg laden
C388:  20 5E C1  JSR  $C15E      Adresse der aktuellen Zeile setzen
C38B:  A4 E6     LDY  * $E6      Linke Fenstergrenze in Y-Reg laden
C38D:  E4 EB     CPX  * $EB      vergleiche m. aktueller Cursor Zeile
C38F:  F0 0F     BEQ  $C3A0      Cursor-Zeile ist untere Grenze
C391:  CA        DEX             Zeile um eins erniedrigen, um dann
C392:  20 76 CB  JSR  $CB76      das Zeilenüberlaufbit zu testen
C395:  E8        INX             wieder auf aktuelle Zeile
C396:  20 83 CB  JSR  $CB83      Zeilenüberlaufbit setzen/löschen
C399:  CA        DEX             wieder auf vorangehende Zeile
C39A:  20 0D C4  JSR  $C40D      MOVLIN: Kopieren einer Fensterzeile
C39D:  4C 88 C3  JMP  $C388      Zurück in die Schleife
C3A0:  20 A5 C4  JSR  $C4A5      Lösche Zeile X
C3A3:  4C 93 CB  JMP  $CB93      Setzen des Zeilenüberlaufbits

******************************     Aufwärts scrollen

C3A6:  A6 E5     LDX  * $E5      Obere Fenstergrenze in X-Reg laden
C3A8:  E8        INX             und um eine Zeile erhöhen
C3A9:  20 76 CB  JSR  $CB76      Teste das Zeilenüberlaufbit
C3AC:  90 0A     BCC  $C3B8      Zeile ist ohne Überlauf
C3AE:  E4 E4     CPX  * $E4      vergleiche mit unterer Fenstergrenze
C3B0:  90 F6     BCC  $C3A8      Grenze noch nicht erreicht
C3B2:  A6 E5     LDX  * $E5      Obere Fenstergrenze in X-Reg laden
C3B4:  E8        INX             und um 1 erhöhen
C3B5:  20 85 CB  JSR  $CB85      Zeilenüberlaufbit setzen
```

```
C3B8:  C6 EB     DEC  * $EB     aktuelle Cur. Zeile um 1 vermindern
C3BA:  24 E8     BIT  * $E8     Teste das 7. Bit von Eingabestart-
C3BC:  30 02     BMI  $C3C0     Zeile, und springe, wenn gesetzt
C3BE:  C6 E8     DEC  * $E8     sonst erniedrige die Eingabezeile
C3C0:  A6 E5     LDX  * $E5     Obere Fenstergrenze in X-Reg laden
C3C2:  E4 DF     CPX  * $DF     mit Cursor-Zeile vergleichen
C3C4:  B0 02     BCS  $C3C8     wenn >= obere Grenze, dann Sprung
C3C6:  C6 DF     DEC  * $DF     Erniedrige Cursorzeile
C3C8:  20 DC C3  JSR  $C3DC     Restbildschirm verschieben
C3CB:  A6 E5     LDX  * $E5     Obere Fenstergrenze in X-Reg laden
C3CD:  20 76 CB  JSR  $CB76     Teste Zeilenüberlaufbit
C3D0:  08        PHP            Rette Flags auf Stack
C3D1:  20 85 CB  JSR  $CB85     Lösche Zeilenüberlaufbit akt. Zeile
C3D4:  28        PLP            und hole Flags wieder zurück
C3D5:  90 04     BCC  $C3DB     Wenn Carry gelöscht, dann Ende
C3D7:  24 F8     BIT  * $F8     sonst teste Scroll-Flag
C3D9:  30 CB     BMI  $C3A6     Bit 7 gesetzt, dann weiter scrollen
C3DB:  60        RTS            Rücksprung aus dem Unterprogramm

******************************  Zeile X löschen (mit Verschieben)

C3DC:  20 5E C1  JSR  $C15E     Zeile X anmelden
C3DF:  A4 E6     LDY  * $E6     Linke Fenstergrenze in Y-Reg laden
C3E1:  E4 E4     CPX  * $E4     vergleiche mit unterer Fenstergrenze
C3E3:  B0 0F     BCS  $C3F4     Grenze ist erreicht
C3E5:  E8        INX            Zeiger zeigt auf Folgezeile
C3E6:  20 76 CB  JSR  $CB76     Teste Zeilenüberlaufbit
C3E9:  CA        DEX            Wieder auf aktuelle Zeile zeigen
C3EA:  20 83 CB  JSR  $CB83     Setzen/Löschen Zeilenüberlaufbit
C3ED:  E8        INX            Wieder auf Folgezeile zeigen
C3EE:  20 0D C4  JSR  $C40D     MOVLIN: Fensterzeile kopieren
C3F1:  4C DC C3  JMP  $C3DC     Nächste Zeile kopieren

******************************  Commodore-Taste abfragen - warten

C3F4:  20 A5 C4  JSR  $C4A5     Lösche Zeile X
C3F7:  A9 7F     LDA  * $7F     Flag für RUN/DIRECT-Modus
C3F9:  8D 00 DC  STA  $DC00     Nach PRA in CIA zur Tastaturabfrage
C3FC:  AD 01 DC  LDA  $DC01     Hole Tastaturmatrix
C3FF:  C9 DF     CMP  # $DF     Commodore-Taste gedrückt?
C401:  D0 09     BNE  $C40C     Wenn nicht gedrückt, dann Ende
C403:  A0 00     LDY  # $00     Commodore-Taste ist gedrückt
C405:  EA        NOP            Beim Scrollen wird nun eine Warte-
C406:  CA        DEX            schleife durchlaufen, um die
C407:  D0 FC     BNE  $C405     Ausgabe etwas zu verzögern
C409:  88        DEY            Die Schleife zählt von 0 bis
```

```
C40A:  D0 F9     BNE  $C405     65536 und hört dann auf
C40C:  60        RTS            Rücksprung aus dem Unterprogramm

******************************  MOVLIN: Kopieren einer Fensterzeile

C40D:  24 D7     BIT  * $D7     Teste 40/80-Zeichen-Modus
C40F:  30 25     BMI  $C436     Springe bei 80-Zeichen-Modus
C411:  BD 33 C0  LDA  $C033,X   Hole Lo-Byte der akt. Zeile
C414:  85 DC     STA  * $DC     Das Lo-Byte in $DA und $DC
C416:  85 DA     STA  * $DA     zwischenspeichern
C418:  BD 4C C0  LDA  $C04C,X   Hole das Hi-Byte der akt. Adresse
C41B:  29 03     AND  # $03     Ausmaskieren der Bits 2-7
C41D:  0D 3B 0A  ORA  $0A3B     und mit Video-Basisadresse ver-
C420:  85 DB     STA  * $DB     knüpfen und abspeichern
C422:  29 03     AND  # $03     Bits 0 und 1 mit Basisadresse
C424:  09 D8     ORA  # $D8     Farb-RAM verknüpfen
C426:  85 DD     STA  * $DD     und als Hi-Byte ablegen
C428:  B1 DA     LDA  ($DA),Y   Hole Quellzeichen und speichere
C42A:  91 E0     STA  ($E0),Y   es an Zieladresse ab. Danach
C42C:  B1 DC     LDA  ($DC),Y   hole Quellfarbe und speichere
C42E:  91 E2     STA  ($E2),Y   auch diese in Zieladresse ab
C430:  C4 E7     CPY  * $E7     vergleiche mit rechter Fenstergrenze
C432:  C8        INY            Erhöhe den Spaltenzeiger
C433:  90 F3     BCC  $C428     Springe, wenn noch nicht Ende
C435:  60        RTS            Rücksprung aus dem Unterprogramm

******************************  Kopieren einer Zeile in 80-Zeichen

C436:  8E 31 0A  STX  $0A31     Zeilennr. zwischenspeichern
C439:  8C 32 0A  STY  $0A32     Spalte zwischenspeichern
C43C:  A2 18     LDX  # $18     Register 24 enthält COPY-Bit
C43E:  20 DA CD  JSR  $CDDA     und Registerwert holen
C441:  09 80     ORA  # $80     setze COPY-Bit und speichere
C443:  20 CC CD  JSR  $CDCC     Register wieder zurück in VDC
C446:  20 E6 CD  JSR  $CDE6     Setze Update-Adresse auf akt. Pos.
C449:  AE 31 0A  LDX  $0A31     Hole zu kopierende Zeile
C44C:  BD 33 C0  LDA  $C033,X   Lo-Byte der zu kopierenden Zeile
C44F:  0A        ASL  A         mal 2, da 80 Zeichen
C450:  85 DA     STA  * $DA     und Lo-Byte abspeichern
C452:  BD 4C C0  LDA  $C04C,X   Hole Hi-Byte der zu kopierenden
C455:  29 03     AND  # $03     Zeile u. maskiere Bits 3-7 aus
C457:  2A        ROL  A         Hole Übertrag rein (*2)
C458:  0D 2E 0A  ORA  $0A2E     Video-RAM-Basis hinzuaddieren
C45B:  85 DB     STA  * $DB     und als Hi-Byte merken
C45D:  A2 20     LDX  # $20     Blockstartadresse Hi
C45F:  18        CLC            Lösche Carry für Addition
C460:  98        TYA            Hole Spalte in den Akku und
C461:  65 DA     ADC  * $DA     addiere Lo-Byte
```

```
C463:  85 DA     STA  * $DA    Startadresse+Spalte nach Lo-Byte
C465:  A9 00     LDA  # $00    Lade Akku mit Null, um den Übertrag
C467:  65 DB     ADC  * $DB    zum Hi-Byte hinzuzuaddieren
C469:  85 DB     STA  * $DB    und speicher Hi-Byte erneut ab
C46B:  20 CC CD  JSR  $CDCC    gleichzeitig als Block-Start-Adr.
C46E:  E8        INX           Zeiger zeigt auf Block-St.-Ad. Lo
C46F:  A5 DA     LDA  * $DA    hole das Lo-Byte der Zieladresse
C471:  20 CC CD  JSR  $CDCC    und VDC mitteilen
C474:  38        SEC           setze Carry für Subtraktion
C475:  A6 E7     LDX  * $E7    Rechte Fenstergrenze in X-Reg laden
C477:  E8        INX           Plus eins
C478:  8A        TXA           und dann nach Akku
C479:  ED 32 0A  SBC  $0A32    Subtrahiere die akt. Spalte
C47C:  8D 32 0A  STA  $0A32    und als Anzahl merken
C47F:  A2 1E     LDX  # $1E    VDC-Word-Count-Register
C481:  20 CC CD  JSR  $CDCC    anmelden und kopieren beginnen
C484:  A2 20     LDX  # $20    Block-Start-Adresse Hi
C486:  A5 DB     LDA  * $DB    Hole Hi-Byte der Quelladresse
C488:  29 07     AND  # $07    Maskiere Bits 3-7 aus
C48A:  0D 2F 0A  ORA  $0A2F    und addiere Attribut-RAM
C48D:  20 CC CD  JSR  $CDCC    anmelden und setzen des Registers
C490:  E8        INX           Zeiger auf Block-Start-Adresse Lo
C491:  A5 DA     LDA  * $DA    Hole Quelladresse Lo
C493:  20 CC CD  JSR  $CDCC    anmelden und setzen
C496:  20 F9 CD  JSR  $CDF9    Setze Update-Adresse für Attribut
C499:  AD 32 0A  LDA  $0A32    Hole Anzahl zu kopierender Zeichen
C49C:  A2 1E     LDX  # $1E    Register 31 ist Word-Count-Register
C49E:  20 CC CD  JSR  $CDCC    anmelden und kopieren
C4A1:  AE 31 0A  LDX  $0A31    Hole aktuelle Zeile zurück
C4A4:  60        RTS           Rücksprung aus dem Unterprogramm

*******************************  Lösche (Zeile X) 40 Zeichen

C4A5:  A4 E6     LDY  * $E6    Linke Fenstergrenze in Y-Reg laden
C4A7:  20 85 CB  JSR  $CB85    Lösche Zeilenüberlaufbit
C4AA:  20 5E C1  JSR  $C15E    Startadresse von Zeile X holen
C4AD:  24 D7     BIT  * $D7    Teste 40/80-Zeichen-Modus
C4AF:  30 0F     BMI  $C4C0    Springe, wenn 80-Zeichen-Modus
C4B1:  88        DEY           Dummy-Dekrement, wird wieder erhöht
C4B2:  C8        INY           Inkrementiere Spaltenzeiger
C4B3:  A9 20     LDA  # $20    Lade Akku mit <Space>
C4B5:  91 E0     STA  ($E0),Y  Speichere Space in Video-RAM
C4B7:  A5 F1     LDA  * $F1    Farbcode für Zeichenausgabe in Akku
C4B9:  91 E2     STA  ($E2),Y  Speichere Farbe in Farb-RAM
C4BB:  C4 E7     CPY  * $E7    vergleiche mit rechter Fenstergrenze
C4BD:  D0 F3     BNE  $C4B2    springe, wenn nicht fertig
C4BF:  60        RTS           Rücksprung aus dem Unterprogramm
```

```
*****************************           Lösche Zeile - 80 Zeichen

C4C0:  8E 31 0A   STX  $0A31    X-Register zwischenspeichern
C4C3:  8C 32 0A   STY  $0A32    Y-Register zwischenspeichern
C4C6:  A2 18      LDX  # $18    Register 24 auswählen
C4C8:  20 DA CD   JSR  $CDDA    Hole aktuellen Wert
C4CB:  29 7F      AND  # $7F    Copy-Bit löschen
C4CD:  20 CC CD   JSR  $CDCC    und neuen Wert wieder ablegen
C4D0:  A2 12      LDX  # $12    Update-Adresse Hi
C4D2:  18         CLC           Lösche Carry für Addition
C4D3:  98         TYA           Hole Spalte nach Akku
C4D4:  65 E0      ADC  * $E0    Addiere Startadresse Lo
C4D6:  48         PHA           Rette Lo-Adresse auf Stack
C4D7:  8D 3C 0A   STA  $0A3C    Sichern des Lo-Bytes
C4DA:  A9 00      LDA  # $00    Lade Akku mit Null, um Übertrag
C4DC:  65 E1      ADC  * $E1    zu Hi-Byte hinzuzuaddieren
C4DE:  8D 3D 0A   STA  $0A3D    Abspeichern des Hi-Byte
C4E1:  20 CC CD   JSR  $CDCC    und ins Register ablegen
C4E4:  E8         INX           Update-Adresse Lo
C4E5:  68         PLA           Hole Lo-Byte von Stack
C4E6:  20 CC CD   JSR  $CDCC    Lo-Byte nach VDC
C4E9:  A9 20      LDA  # $20    Lade Akku mit <Space>
C4EB:  20 CA CD   JSR  $CDCA    Und ins VDC-Data-Register
C4EE:  38         SEC           Setze Carry für Subtraktion
C4EF:  A5 E7      LDA  * $E7    Rechte Fenstergrenze in Akku laden
C4F1:  ED 32 0A   SBC  $0A32    subtrahiere Startspalte
C4F4:  48         PHA           Rette Anzahl auf Akku
C4F5:  F0 14      BEQ  $C50B    Startspalte ist gleich rechter Rand
C4F7:  AA         TAX           Hole Anzahl nach X
C4F8:  38         SEC           Setze Carry für Addition
C4F9:  6D 3C 0A   ADC  $0A3C    Addiere Lo-Byte
C4FC:  8D 3C 0A   STA  $0A3C    und wieder abspeichern
C4FF:  A9 00      LDA  # $00    Lade Akku mit Null, um den Übertrag
C501:  6D 3D 0A   ADC  $0A3D    zum Hi-Byte zu addieren
C504:  8D 3D 0A   STA  $0A3D    Hi-Byte auch abspeichern
C507:  8A         TXA           Hole Anzahl Zeichen in den Akku
C508:  20 3E C5   JSR  $C53E    Akku ins Wordcount-Register
C50B:  A2 12      LDX  # $12    Update-Adresse Hi
C50D:  18         CLC           Lösche Carry für Addition
C50E:  98         TYA           Hole Spalte in den Akku
C50F:  65 E2      ADC  * $E2    und addiere Lo-Byte Attribut
C511:  48         PHA           Rette Lo-Byte auf Stack
C512:  A9 00      LDA  # $00    Lade Akku mit Null, um den
C514:  65 E3      ADC  * $E3    Übertrag hinzuzuaddieren
C516:  20 CC CD   JSR  $CDCC    und schreibe das Hi-Byte ins
C519:  E8         INX           Register. Update-Adresse Lo
C51A:  68         PLA           Hole Lo-Byte von Stack
C51B:  20 CC CD   JSR  $CDCC    und ins Register schreiben
```

```
C51E:  AD 3D 0A   LDA  $0A3D      Hole Hi-Byte Zieladresse
C521:  29 07      AND  # $07      Ausmaskieren der Bits 4-7
C523:  0D 2F 0A   ORA  $0A2F      und mit der Video-Basisadresse
C526:  8D 3D 0A   STA  $0A3D      verknüpfen und abspeichern
C529:  A5 F1      LDA  * $F1      Farbcode für Zeichenausgabe in Akku
C52B:  29 8F      AND  # $8F      Nur Farbe und ALT-Bit sind relevant
C52D:  20 CA CD   JSR  $CDCA      Hole Registerinhalt von DATA-REG
C530:  68         PLA             Hole Anzahl von Stack
C531:  F0 03      BEQ  $C536      Wenn null, dann Sprung
C533:  20 3E C5   JSR  $C53E      Gib Farbe aus
C536:  AE 31 0A   LDX  $0A31      X-Register zurückholen
C539:  A4 E7      LDY  * $E7      Rechte Fenstergrenze in Y-Reg laden
C53B:  60         RTS             Rücksprung aus dem Unterprogramm

******************************    Schreibe Akku mal Zeichen an Update
                                  Register

C53C:  A9 01      LDA  # $01      Lade Zähler mit 1
C53E:  A2 1E      LDX  # $1E      Wordcount-Register auswählen
C540:  20 CC CD   JSR  $CDCC      und Wert übermitteln
C543:  2C 00 D6   BIT  $D600      Teste Statusbit
C546:  10 FB      BPL  $C543      und warte, bis fertig
C548:  A2 12      LDX  # $12      Update-Adresse Hi
C54A:  20 DA CD   JSR  $CDDA      Hole aktuellen Wert
C54D:  CD 3D 0A   CMP  $0A3D      Vergleiche mit Hi-Byte Zieladresse
C550:  90 EA      BCC  $C53C      stimmt nicht: Fehler korrigieren
C552:  A2 13      LDX  # $13      Update-Adresse Lo
C554:  20 DA CD   JSR  $CDDA      Aktuellen Wert holen
C557:  CD 3C 0A   CMP  $0A3C      Vergleiche mit Zieladresse Lo
C55A:  90 E0      BCC  $C53C      stimmt nicht: Fehler korrigieren
C55C:  60         RTS             Rücksprung aus dem Unterprogramm

******************************    Prüfen der Tastatur Matrix

C55D:  A5 01      LDA  * $01      Hole Bit 6 aus Z-Page Datenregister
C55F:  29 40      AND  # $40      Prozessorport. Bit 6 zeigt, ob DIN
C561:  49 40      EOR  # $40      oder ASCII Zeichensatz gewählt ist.
C563:  4A         LSR  A          Invertiere Bit 6 und bringe es an
C564:  4A         LSR  A          Position Bit 4. Shift Flag zurück-
C565:  85 D3      STA  * $D3      setzen und DIN/ASCII Mode sichern.
C567:  A0 58      LDY  # $58      Code für "Keine Taste" in Z-Page
C569:  84 D4      STY  * $D4      Zeiger für gedrückte Taste ablegen
C56B:  A9 00      LDA  # $00      Prüfwert für die Matrixzeilen
C56D:  8D 00 DC   STA  $DC00      Zuständig für Matrixzeilen 1 - 8
C570:  8D 2F D0   STA  $D02F      Zuständig für Matrixzeilen 9 -11
C573:  AE 01 DC   LDX  $DC01      Port B = Eingang der Matrixspalten
C576:  E0 FF      CPX  # $FF      Prüfe, ob eine Taste gedrückt ist
C578:  D0 03      BNE  $C57D      Überprüfen, welche Taste gedrückt
```

```
C57A:  4C 97 C6   JMP  $C697      Keine Taste, dann weiter
C57D:  A8         TAY             Disp.-Zähler a. Tastatur-Tab.-Anfang
C57E:  AD 3E 03   LDA  $033E      Adresse Lo der Tastatur-Decodier-
C581:  85 CC      STA  * $CC      tabelle 1a in Z-Page kopieren
C583:  AD 3F 03   LDA  $033F      Adresse Hi der Tastatur-Decodier-
C586:  85 CD      STA  * $CD      tabelle 1a in Z-Page kopieren
C588:  A9 FF      LDA  # $FF      Testwert für Tastaturmatrix
C58A:  8D 2F D0   STA  $D02F      Testzeilen 9 -11 auf Hi setzen
C58D:  2A         ROL  A          Bitposition der Testzeile auf 0
C58E:  24 D3      BIT  * $D3      Zeiger, ob 1-8 oder 9-11 zu testen
C590:  30 05      BMI  $C597      Wenn Zeile 9-11 zu testen, dann Skip
C592:  8D 00 DC   STA  $DC00      Testwert in Port A (Matrixzeile 1-8)
C595:  10 03      BPL  $C59A      Skip Test der Matrixzeilen 9-11
C597:  8D 2F D0   STA  $D02F      Port A* (Matrixzeilen 9-11) testen
C59A:  A2 08      LDX  # $08      Zähler für 8 Matrixspalten setzen
C59C:  48         PHA             Zeilentestwert auf Akku sichern
C59D:  AD 01 DC   LDA  $DC01      Port B (Ausgabe der Matrixspalten)
C5A0:  CD 01 DC   CMP  $DC01      mit Port B vergleichen (Tastatur
C5A3:  D0 F8      BNE  $C59D      entprellen) und warten
C5A5:  4A         LSR  A          Den Ausgabewert der Matrixspalten
C5A6:  B0 17      BCS  $C5BF      bitweise testen. C=1 ist keine Taste
C5A8:  48         PHA             Matrixspalten Ausgabewert sichern
C5A9:  B1 CC      LDA  ($CC),Y    Tastencode aus Tastaturtabelle holen
C5AB:  C9 08      CMP  # $08      Tastencode 8 ist ALT Taste
C5AD:  F0 08      BEQ  $C5B7      zur entsprechenden Auswertung
C5AF:  C9 05      CMP  # $05      Prüfe, ob es Code f. Shift, C- oder
C5B1:  B0 09      BCS  $C5BC      Ctrl ist. Nein, dann weiter
C5B3:  C9 03      CMP  # $03      Ist es der Code für BREAK Taste?
C5B5:  F0 05      BEQ  $C5BC      Ja, dann weiter für Break Taste
C5B7:  05 D3      ORA  * $D3      Z-Page Zeiger für Shift Muster
C5B9:  85 D3      STA  * $D3      mit dem Akku verknüpfen
C5BB:  2C         .Byte  $2C      Skip nach $C5BE
C5BC:  84 D4      STY  * $D4      In Z-Page Flag f. Tastencode ablegen
C5BE:  68         PLA             Matrixspalten Testwert zurückholen
C5BF:  C8         INY             Tastatur Tab. Disp. Zähler + 1
C5C0:  CA         DEX             Matrixspalten Schleifenzähler - 1
C5C1:  D0 E2      BNE  $C5A5      Schleifen, bis alle Spalten getestet
C5C3:  C0 59      CPY  # $59      Sind alle Zeilen + Spalten getestet?
C5C5:  B0 10      BCS  $C5D7      Ja, dann Tastendruck auswerten
C5C7:  68         PLA             Zeilentestwert vom Stack holen
C5C8:  38         SEC             Carry Flag für Verschiebung des
C5C9:  2A         ROL  A          Zeilentestwertes setzen
C5CA:  B0 C2      BCS  $C58E      Weiter im Test der Matrixzeilen 1-8
C5CC:  8D 00 DC   STA  $DC00      Port A Testwert auf Hi ($FF) setzen
C5CF:  26 D3      ROL  * $D3      Bit 7 im Shiftmusterflag einblenden,
C5D1:  38         SEC             da nun die verbleibenden Matrixzeilen
C5D2:  66 D3      ROR  # $D3      9-11 über Port A* zu testen sind.
C5D4:  2A         ROL  A          Bit f. Matrixzeilentest 9-11 löschen
```

```
C5D5:  D0 B7      BNE  $C58E       Sprung: Nächste Matrixzeile testen

******************************     Auswertung des Tastaturergebnisses

C5D7:  06 D3      ASL  * $D3       Das im Shiftmusterflag gesetzte Bit
C5D9:  46 D3      LSR  * $D3       7 (KZ für Port A* Test) eliminieren
C5DB:  68         PLA              Zeilentestwert vom Stack löschen
C5DC:  A5 D4      LDA  * $D4       Code für gedrückte Taste in Akku
C5DE:  6C 3A 03   JMP  ($033A)     Vektor für Tastaturabfrage   ($C5E1)

******************************     Routine: Tastatur auswerten

C5E1:  C9 57      CMP  # $57       War es die "No Scroll" Taste?
C5E3:  D0 13      BNE  $C5F8       Nein, dann Skip
C5E5:  24 F7      BIT  * $F7       Z-Page Pause Flag Bit 6: 1=disable
C5E7:  70 5A      BVS  $C643       Wenn Pause nicht erlaubt, dann RTS
C5E9:  AD 25 0A   LDA  $0A25       Lade Akku mit letztem Shift-Muster
C5EC:  D0 55      BNE  $C643       Ungleich 0, dann Exit über RTS
C5EE:  A9 0D      LDA  # $0D       Die Bits 0, 1, und 3 des Z-Page
C5F0:  4D 21 0A   EOR  $0A21       Pause Zeigers invertieren und im
C5F3:  8D 21 0A   STA  $0A21       Zero-Page Pause Zeiger ablegen
C5F6:  50 30      BVC  $C628       Tastatur-Repeat-Routine
C5F8:  A5 D3      LDA  * $D3       Hole aktuelles SHIFT-Muster in Akku
C5FA:  F0 55      BEQ  $C651       Kein Shift-Muster, normal auswerten
C5FC:  C9 10      CMP  # $10       War der DIN-Zeichensatz gewählt?
C5FE:  F0 44      BEQ  $C644       Ja, dann zur DIN-Auswertung
C600:  C9 08      CMP  # $08       War ALT-Tastendruck gekennzeichnet?
C602:  F0 42      BEQ  $C646       Ja, dann zur ALT-Auswertung
C604:  29 07      AND  # $07       Bit 3 - 7 aus Shiftmuster ausblenden
C606:  C9 03      CMP  # $03       Wurde C- Shift Umschaltung gewählt?
C608:  D0 25      BNE  $C62F       Nein, Shiftmuster weiter auswerten

******************************     C=/Shift Zeichensatzumschaltung

C60A:  A5 F7      LDA  * $F7       Prüfe Flag für C= Shift Umschaltung
C60C:  30 43      BMI  $C651       Umschaltung verboten,z. REPEAT Rout.
C60E:  AD 25 0A   LDA  $0A25       Hole letztes gesichertes Shiftmuster
C611:  D0 3E      BNE  $C651       Nicht Null, dann zur REPEAT Routine
C613:  24 D7      BIT  * $D7       Prüfe auf 40/80 Zeichen Bildschirm
C615:  10 09      BPL  $C620       Positiv = 40 Zeichen Bildschirm
C617:  A5 F1      LDA  * $F1       Farbcode für Zeichenausgabe in Akku
C619:  49 80      EOR  # $80       Bit 7 des Farbcodes invertieren
C61B:  85 F1      STA  * $F1       Farbcode für Zeichenausgabe sichern
C61D:  4C 28 C6   JMP  $C628       Überspringe VIC Zeichenumschaltung
C620:  AD 2C 0A   LDA  $0A2C       Systemzeiger für Text/Bildschirm
C623:  49 02      EOR  # $02       Basis holen und das Bit 2 dieses
C625:  8D 2C 0A   STA  $0A2C       Zeigers invertieren
C628:  A9 08      LDA  # $08       Den Systemzeiger für das letzte
```

```
C62A:  8D 25 0A   STA  $0A25      Shiftmuster mit 8 initialisieren
C62D:  D0 22      BNE  $C651      Sprung zur Repeat Routine

****************************      Decodiertabelle entsprechend dem
                                  Shiftmuster laden und auswerten

C62F:  0A         ASL  A          Shiftmuster f. Displ. mit 2 multipl.
C630:  C9 08      CMP  # $08      Wurde Shiftmuster für Shift oder C=
C632:  90 12      BCC  $C646      gefunden, dann Decodiertabelle laden
C634:  A9 06      LDA  # $06      Defaultwert für Ctrl Muster in Akku
C636:  A6 D4      LDX  * $D4      Prüfe den Offset auf Decodiertabelle
C638:  E0 0D      CPX  # $0D      Wenn es die 13. Taste war (S-Taste),
C63A:  D0 0A      BNE  $C646      dann Pause Flag setzen, sonst Skip
C63C:  24 F7      BIT  * $F7      Prüfe, ob Pause/Ctrl-S erlaubt ist
C63E:  70 06      BVS  $C646      Nicht erlaubt,Decodiertab. auswerten
C640:  8E 21 0A   STX  $0A21      Pause Flag mit Tastenwert 13 setzen
C643:  60         RTS             Rücksprung aus dem Unterprogramm

****************************      Startadresse der Decodiertabelle
                                  entsprechend Shift Muster setzen

C644:  A9 0A      LDA  # $0A      Defaultwert auf Tabelle 5a setzen
C646:  AA         TAX             Nr. der Decodiertabelle in X-Reg
C647:  BD 3E 03   LDA  $033E,X    Adresse Lo der Decodiertabelle
C64A:  85 CC      STA  * $CC      in Z-Page Speicher kopieren
C64C:  BD 3F 03   LDA  $033F,X    Adresse Hi der Decodiertabelle
C64F:  85 CD      STA  * $CD      in Z-Page Speicher kopieren

****************************      Routine REPEAT
                                  Wiederholung der Tastatur Logik

C651:  A4 D4      LDY  * $D4      Displ. auf Tabellenanfang in Y-Reg
C653:  B1 CC      LDA  ($CC),Y    Akku mit Zeichencode aus Tab. laden
C655:  AA         TAX             und Zeichen in X-Reg sichern
C656:  C4 D5      CPY  * $D5      Mit Zeiger für momentane Taste vgl.
C658:  F0 07      BEQ  $C661      Wenn gleich zur Repeat Prüfung
C65A:  A0 10      LDY  # $10      Zähler f. Tastenwiederholverzögerung
C65C:  8C 24 0A   STY  $0A24      mit $10 initialisieren
C65F:  D0 36      BNE  $C697      Sprung zur Tastendruck Auswertung
C661:  29 7F      AND  # $7F      Bit 7 ausblenden, keine RVS Zeichen
C663:  2C 22 0A   BIT  $0A22      Prüfe Zeiger für Tastenwiederholung
C666:  30 16      BMI  $C67E      Alle Tasten zugelassen ($80), Skip
C668:  70 5A      BVS  $C6C4      Keine Taste erlaubt ($40), Skip
C66A:  C9 7F      CMP  # $7F      Prüfe, ob "Zeichen ungültig"
C66C:  F0 29      BEQ  $C697      Ja, dann Default Abfrage und RTS
C66E:  C9 14      CMP  # $14      War es die <DEL> Taste?
C670:  F0 0C      BEQ  $C67E      Ja, dann Wiederholungsauswertung
C672:  C9 20      CMP  # $20      War es die <Space> Taste?
```

```
C674:  F0 08     BEQ  $C67E     Ja, dann Wiederholungsauswertung
C676:  C9 1D     CMP  # $1D     War es die <Crsr right> Taste?
C678:  F0 04     BEQ  $C67E     Ja, dann Wiederholungsauswertung
C67A:  C9 11     CMP  # $11     War es die <Crsr down> Taste?
C67C:  D0 46     BNE  $C6C4     Nein, Skip Wiederholungsauswertung

*****************************   Tasten Wiederholungsauswertung

C67E:  AC 24 0A  LDY  $0A24     Hole Zähler für Wiederholverzögerung
C681:  F0 05     BEQ  $C688     Zähler = null, dann Skip
C683:  CE 24 0A  DEC  $0A24     Wiederholungsverzögerungszähler -1
C686:  D0 3C     BNE  $C6C4     Nicht null, Default-Abfrage und RTS
C688:  CE 23 0A  DEC  $0A23     Zählgeschwindigkeit für Repeat -1
C68B:  D0 37     BNE  $C6C4     Nicht null, Default-Abfrage und RTS
C68D:  A0 04     LDY  # $04     Zählgeschwindigkeit für Tastenrepeat
C68F:  8C 23 0A  STY  $0A23     mit $04 neu initialisieren
C692:  A4 D0     LDY  * $D0     Offset der Keybuf.-Schlange in Y-Reg
C694:  88        DEY            Wenn mehr als 1 Zeichen im Puffer
C695:  10 2D     BPL  $C6C4     steht, dann Default-Abfrage und RTS

*****************************   Einsprung: Keine Taste gedrückt

C697:  4E 25 0A  LSR  $0A25     Letztes Shiftmuster d. 2 dividieren
C69A:  A4 D4     LDY  * $D4     Displ. auf Decodiertab. Anfang in
C69C:  84 D5     STY  * $D5     Zeiger für momentane Taste kopieren
C69E:  E0 FF     CPX  # $FF     War es der Code für "kein Zeichen"?
C6A0:  F0 22     BEQ  $C6C4     Ja, dann Default Abfrage und RTS
C6A2:  A9 00     LDA  # $00     Bei gültigem Zeichen den Pause/
C6A4:  8D 21 0A  STA  $0A21     Ctrl-S Zeiger zurücksetzen
C6A7:  8A        TXA            Zeichencode in Akku kopieren
C6A8:  A6 D3     LDX  * $D3     hole aktuelles SHIFT-Muster in X-Reg
C6AA:  4C C6 FC  JMP  $FCC6     Zurück zur Kernal-Routine: KEY

*****************************   Tastendruck auswerten und speichern

C6AD:  A2 09     LDX  # $09     Schleifenzähler f.10 Funktionstasten
C6AF:  DD DD C6  CMP  $C6DD,X   Vgl. Akku mit Tastencodetabelle
C6B2:  F0 16     BEQ  $C6CA     Funktionstaste gefunden, auswerten
C6B4:  CA        DEX            Schleifenzähler um 1 vermindern
C6B5:  10 F8     BPL  $C6AF     Schleifen,bis alle Vgl. durchgeführt
C6B7:  A6 D0     LDX  * $D0     Index: Tastaturpuffer Warteschlange
C6B9:  EC 20 0A  CPX  $0A20     Mit maximaler Größe vergleichen
C6BC:  B0 06     BCS  $C6C4     Maximale Größe erreicht, dann Skip
C6BE:  9D 4A 03  STA  $034A,X   Zeichen im Tastaturpuffer ablegen
C6C1:  E8        INX            Den Index der Tastaturpuffer Warte-
C6C2:  86 D0     STX  * $D0     schlange um 1 Zeichen erhöhen
C6C4:  A9 7F     LDA  # $7F     Tastatur-Matrix
C6C6:  8D 00 DC  STA  $DC00     auf Default abfragen
```

```
C6C9:  60          RTS              Rücksprung aus dem Unterprogramm

******************************      Präpariert Tastaturbuffer für KEY

C6CA:  BD 00 10    LDA  $1000,X     Hole Länge von KEY X
C6CD:  85 D1       STA  * $D1       und in KEY-Zeichen-Zähler
C6CF:  A9 00       LDA  # $00       Es wird nun die Position des KEYs
C6D1:  CA          DEX              in der Gesamttabelle ermittelt
C6D2:  30 06       BMI  $C6DA       Wenn alle Längen addiert, dann Ende
C6D4:  18          CLC              sonst lösche Carry für Addition
C6D5:  7D 00 10    ADC  $1000,X     Addiere Länge von KEY X
C6D8:  90 F7       BCC  $C6D1       Wenn kein Überlauf, dann weiter
C6DA:  85 D2       STA  * $D2       sonst speichere Pointer ab
C6DC:  60          RTS              Rücksprung aus dem Unterprogramm

******************************      Tastencodes der 10 Funktionstasten

C6DD:  85 89                        F1    F2
C6DF:  86 8A                        F3    F4
C6E1:  87 8B                        F5    F6
C6E3:  88 8C                        F7    F8
C6E5:  83                           F9    (Shift-Run)
C6E6:  84                           F10   (Help-Taste)

******************************      VIC Cursor blinken lassen

C6E7:  24 D7       BIT  * $D7       Teste auf 40/80-Zeichen
C6E9:  30 41       BMI  $C72C       Wenn 80 Zeichen, dann Ende
C6EB:  AD 27 0A    LDA  $0A27       Hole VIC-Cursor-Modus
C6EE:  D0 3C       BNE  $C72C       Ist ausgeschaltet, dann Ende
C6F0:  CE 28 0A    DEC  $0A28       sonst erniedrige den Blinkzähler
C6F3:  D0 37       BNE  $C72C       Wenn noch nicht null, dann Ende
C6F5:  AD 26 0A    LDA  $0A26       Hole VIC-Cursor-Modus
C6F8:  29 C0       AND  # $C0       Maskiere Bits 0-5 aus
C6FA:  C9 C0       CMP  # $C0       Cursor starr oder ausgeschaltet?
C6FC:  F0 2E       BEQ  $C72C       Wenn ja, dann Ende
C6FE:  A9 14       LDA  # $14       Setze den VIC-Cursor-Blink-Zähler
C700:  8D 28 0A    STA  $0A28       auf $14=20
C703:  A4 EC       LDY  * $EC       Hole aktuelle Cursor Spalte in Y-Reg
C705:  AE 2A 0A    LDX  $0A2A       Hole Farbe an Cursorp. zum Blinken
C708:  B1 E0       LDA  ($E0),Y     Hole Zeichen an akt. Spalte
C70A:  2C 26 0A    BIT  $0A26       Teste VIC-Cursor-Modus
C70D:  30 10       BMI  $C71F       Zeichen wieder normal
C70F:  8D 29 0A    STA  $0A29       Zeichen an Cursorpos. vor Blinken
C712:  20 7C C1    JSR  $C17C       Farb-RAM-Adresse setzen
C715:  B1 E2       LDA  ($E2),Y     Hole Farbe an Cursorposition
C717:  8D 2A 0A    STA  $0A2A       als Farbe vor Blinken merken
C71A:  A6 F1       LDX  * $F1       Farbcode für Zeichenausgabe in X-Reg
```

```
C71C:  AD 29 0A   LDA  $0A29      Zeichen an Cursorpos. vor Blinken
C71F:  49 80      EOR  # $80      Invertiere das Negativ-Bit
C721:  20 40 CC   JSR  $CC40      Abspeichern von Zeichen und Farbe
C724:  AD 26 0A   LDA  $0A26      Hole VIC-Cursor-Modus
C727:  49 80      EOR  # $80      Negiere den Blinkzustand
C729:  8D 26 0A   STA  $0A26      und wieder abspeichern
C72C:  60         RTS             Rücksprung aus dem Unterprogramm

******************************    BSOUT Einsprung f. Bildschirmausgabe

C72D:  85 EF      STA  * $EF      Auszugebendes Zeichen in Z-P. retten
C72F:  48         PHA             Akku Inhalt auf Stack retten
C730:  8A         TXA             X-Reg Inhalt über Akku
C731:  48         PHA             auf Stack retten
C732:  98         TYA             Y-Reg Inhalt über Akku
C733:  48         PHA             auf Stack retten
C734:  AD 21 0A   LDA  $0A21      Prüfe d. Inhalt des Z-P. Pause Flags
C737:  D0 FB      BNE  $C734      Warten, bis Flag Wert 0 ist
C739:  85 D6      STA  * $D6      Input/Get Flag über Tastatur löschen
C73B:  A9 C3      LDA  # $C3      Hi-Byte der Fortsetzung auf Stack,
C73D:  48         PHA             um die Routine dann mit RTS anzu-
C73E:  A9 0B      LDA  # $0B      springen. Jetzt auch noch das
C740:  48         PHA             Byte der Fortsetzung auf Stack
C741:  A4 EC      LDY  * $EC      Hole aktuelle Cursor Spalte in Y-Reg
C743:  A5 EF      LDA  * $EF      Hole auszugebendes Zeichen aus ZwSp.
C745:  C9 0D      CMP  # $0D      Ist es ein Wagenrücklauf <Cr> ?
C747:  F0 26      BEQ  $C76F      Ja, dann <Cr> ausgeben
C749:  C9 8D      CMP  # $8D      Ist es ein Shift/Wagenrücklauf ?
C74B:  F0 22      BEQ  $C76F      Ja, dann <Sh/Cr> ausgeben
C74D:  A6 F0      LDX  * $F0      Hole Wert des vorherigen Zeichens
C74F:  E0 1B      CPX  # $1B      War es <Esc>, dann Zeichen als Esc
C751:  D0 03      BNE  $C756      Sequenz behandeln, sonst nach $C756
C753:  4C BE C9   JMP  $C9BE      Zur Auswertung von Escape Sequenzen
C756:  AA         TAX             Auszugebendes Zeichen nach X-Reg
C757:  10 03      BPL  $C75C      Ist es ein Zeichen von 0 - 127 ?
C759:  4C 02 C8   JMP  $C802      Nein, dann Auswertung: Erw. ASCII
C75C:  C9 20      CMP  # $20      Ist auszugebendes Zeichen < Blank?)
C75E:  90 56      BCC  $C7B6      Ja, dann zur Auswert. v. Steuercodes
C760:  C9 60      CMP  # $60      Ist es ein Buchstabenzeichen ?
C762:  90 03      BCC  $C767      Ja, dann zur Buchstabenausgabe
C764:  29 DF      AND  # $DF      Bit 5 ausmaskieren
C766:  .Byte $2C                  Ergibt Scheinbefehl : BIT  $3F29
                                  und Skip nach $C769
```

```
******************************      Buchstaben ausgeben

C767:  29 3F      AND  # $3F      Bit 6/7 des Zeichens ausmaskieren
C769:  20 FF C2   JSR  $C2FF      Teste auf Anführungszeichen
C76C:  4C 22 C3   JMP  $C322      Zeichen ausgeben

******************************      <Carriage Return> - Neue Zeile

C76F:  20 C3 CB   JSR  $CBC3      Suche Ende der Eingabezeile
C772:  E8         INX             Lösche das Zeilenüberlaufbit
C773:  20 85 CB   JSR  $CB85      der Folgezeile
C776:  A4 E6      LDY  * $E6      Linke Fenstergrenze in Y-Reg laden
C778:  84 EC      STY  * $EC      sichern der aktuellen Cursor Spalte
C77A:  20 63 C3   JSR  $C363      Zeilenvorschub ausführen

******************************      Rücksetzen von Quote/Insert/RVS

C77D:  A5 F1      LDA  * $F1      Farbcode für Zeichenausgabe in Akku
C77F:  29 CF      AND  # $CF      Revers und Flash aus bei VDC
C781:  85 F1      STA  * $F1      Farbcode für Zeichenausgabe sichern
C783:  A9 00      LDA  # $00      Lade Akku mit Null für aus
C785:  85 F5      STA  * $F5      und lösche die Bits Insert-Mode
C787:  85 F3      STA  * $F3      RVS-Flag
C789:  85 F4      STA  * $F4      Quote-Mode-Flag
C78B:  60         RTS             Rücksprung aus dem Unterprogramm

******************************      Controlcodes

C78C:  02         .Byte  $02      2 = Unterstreichen ein
C78D:  07         .Byte  $07      7 = Bell
C78E:  09         .Byte  $09      9 = Tab anspringen
C78F:  0A         .Byte  $0A      A = Zeilenvorschub
C790:  0B         .Byte  $0B      B = Sperre <Shift>/<Commodore>
C791:  0C         .Byte  $0C      C = Freigeben von <Sh>/<C=>
C792:  0E         .Byte  $0E      E = Lower Case
C793:  0F         .Byte  $0f      F = Blinken ein
C794:  11         .Byte  $11      11= Cursor Up
C795:  12         .Byte  $12      12= Revers ein
C796:  13         .Byte  $13      13= Home
C797:  14         .Byte  $14      14= Delete
C798:  18         .Byte  $18      18= Tab setzen/löschen
C799:  1D         .Byte  $1d      1D= Cursor Right

******************************      Adressen der Routinen, die die
                                    Controlcodes ausführen (-1)
                                    werden mit RTS angesprungen.

C79A:  C6 C8      $C8C6           Unterstreichen ein
```

```
C79C:  8D C9        $C98D          Tab anspringen
C79E:  4E C9        $C94E          Bell
C7A0:  B0 C9        $C9B0          Zeilenvorschub
C7A2:  A5 C8        $C8A5          Sperren <Sh>/<C=>
C7A4:  AB C8        $C8AB          Freigeben <Sh>/<C=>
C7A6:  7F C8        $C87F          Lower Case
C7A8:  D4 C8        $C8D4          Blinken ein
C7AA:  59 C8        $C859          Cursor Up
C7AC:  C1 C8        $C8C1          Revers ein
C7AE:  B2 C8        $C8B2          Home
C7B0:  1A C9        $C91A          Delete
C7B2:  60 C9        $C960          Tab setzen/löschen
C7B4:  53 C8        $C853          Cursor Rechts

******************************     Controlcodes ausführen

C7B6:  6C 34 03   JMP  ($0334)     Vektor Zeichenausgabe mit Ctrl
C7B9:  C9 1B      CMP  # $1B       ist Zeichen <ESC>?
C7BB:  F0 38      BEQ  $C7F5       Ja, dann Ende
C7BD:  A6 F5      LDX  * $F5       Insert-Modus gesetzt?
C7BF:  D0 08      BNE  $C7C9       Ja, dann Zeichen revers ausgeben
C7C1:  C9 14      CMP  # $14       Ist das Zeichen <Delete>?
C7C3:  F0 0B      BEQ  $C7D0       Dann ausführen
C7C5:  A6 F4      LDX  * $F4       Ist das Quote-Mode-Flag gesetzt?
C7C7:  F0 07      BEQ  $C7D0       Wenn ja, dann Zeichen revers
C7C9:  A2 00      LDX  # $00       Das zuletzt gedruckte Zeichen
C7CB:  86 EF      STX  * $EF       in Zeropage löschen
C7CD:  4C 26 C3   JMP  $C326       und Zeichen revers ausgeben

******************************     Vergleiche A mit mögl. Ctrl.Codes

C7D0:  A2 0D      LDX  # $0D       X ist der Zähler für Control-Codes
C7D2:  DD 8C C7   CMP  $C78C,X     vergleiche mit Tabelle
C7D5:  F0 1F      BEQ  $C7F6       Gefunden? Dann Sprung zum Ausführen
C7D7:  CA         DEX              sonst erniedrige den Zähler und
C7D8:  10 F8      BPL  $C7D2       vergleiche mit nächsten Wert
C7DA:  A2 0F      LDX  # $0F       Nun noch mit den 16 möglichen
C7DC:  DD 4C CE   CMP  $CE4C,X     Codes zum Wechseln der Farbe
C7DF:  F0 04      BEQ  $C7E5       vergleichen. Sprung, wenn gefunden
C7E1:  CA         DEX              sonst erniedrige Zähler und
C7E2:  10 F8      BPL  $C7DC       vergleiche mit nächstem Wert
C7E4:  60         RTS              Rücksprung aus dem Unterprogramm

******************************     Farbe setzen - 40 Zeichen

C7E5:  24 D7      BIT  * $D7       Teste 40/80-Zeichen-Modus
C7E7:  30 03      BMI  $C7EC       Bei 80-Zeichen-Modus springen
C7E9:  86 F1      STX  * $F1       Farbcode für Zeichenausgabe sichern
```

```
C7EB:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   Farbe setzen - 80-Zeichen-Modus

C7EC:  A5 F1     LDA  * $F1      Farbcode für Zeichenausgabe in Akku
C7EE:  29 F0     AND  # $F0      Untere Tetrade (B. 0-3) ausmaskieren
C7F0:  1D 5C CE  ORA  $CE5C,X    mit Farbcodetabelle verknüpfen
C7F3:  85 F1     STA  * $F1      Farbcode für Zeichenausgabe sichern
C7F5:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   Controlcodes ausführen

C7F6:  8A        TXA             Zeiger nach Akku und dann
C7F7:  0A        ASL             mit zwei multiplizieren, da ein
C7F8:  AA        TAX             16-Bit-Wert geholt wird.
C7F9:  BD 9B C7  LDA  $C79B,X    Hole Lo-Byte der Startadresse
C7FC:  48        PHA             in Akku und hole
C7FD:  BD 9A C7  LDA  $C79A,X    Hi-Byte der Startadresse auch
C800:  48        PHA             in Akku. Wird dann mit


C801:  60        RTS             RTS angesprungen

******************************   Erweiterten ASCII analysieren

C802:  6C 36 03  JMP  ($0336)    Vektor Zeichenausgabe mit Shift
C805:  29 7F     AND  # $7F      Bit 7 ausblenden, nicht geshifted
C807:  C9 20     CMP  # $20      Vergleiche mit <Space>
C809:  90 09     BCC  $C814      Kleiner als 32
C80B:  C9 7F     CMP  # $7F      Ist es ASCII-Code 127?
C80D:  D0 02     BNE  $C811      Falls nicht, dann springe
C80F:  A9 5E     LDA  # $5E      ASCII-Code für Pfeil-nach-oben
C811:  4C 20 C3  JMP  $C320      und ausgeben
C814:  A6 F4     LDX  * $F4      Hole Quote-Mode-Flag
C816:  F0 05     BEQ  $C81D      Springe, wenn nicht gesetzt
C818:  09 40     ORA  # $40      sonst setze Bit 6
C81A:  4C 26 C3  JMP  $C326      als Reverszeichen ausgeben
C81D:  C9 14     CMP  # $14      Ist das Zeichen <INSERT>?
C81F:  D0 03     BNE  $C824      Springe, wenn nicht <INSERT>
C821:  4C E3 C8  JMP  $C8E3      Sonst führe <INSERT> aus
C824:  A6 F5     LDX  * $F5      Hole Insert-Mode-Flag
C826:  D0 F0     BNE  $C818      Wenn gesetzt, dann wie bei Quote
C828:  C9 11     CMP  # $11      Vergleiche mit Cursor-Hoch
C82A:  F0 3B     BEQ  $C867      Springe, wenn Cursor-Up
C82C:  C9 1D     CMP  # $1D      Cursor-Links?
C82E:  F0 45     BEQ  $C875      Wenn ja, dann führe aus
C830:  C9 0E     CMP  # $0E      Vergleiche, ob UPPERCASE
C832:  F0 5E     BEQ  $C892      Springe zur Ausführung
```

```
C834:  C9 12      CMP  # $12     Revers off?
C836:  D0 03      BNE  $C83B     Nein, dann überspringe
C838:  4C BF C8   JMP  $C8BF     Sonst lösche RVS-Modus
C83B:  C9 02      CMP  # $02     Unterstreichen einschalten?
C83D:  D0 03      BNE  $C842     Wenn nicht, dann überspringe
C83F:  4C CE C8   JMP  $C8CE     Sonst setze Unterstreich-Modus
C842:  C9 0F      CMP  # $0F     Blink-Modus ausschalten?
C844:  D0 03      BNE  $C849     Überspringe, wenn nicht
C846:  4C DC C8   JMP  $C8DC     sonst lösche Blink-Modus
C849:  C9 13      CMP  # $13     Ist es <CLR/HOME>?
C84B:  D0 03      BNE  $C850     Überspringe, wenn nein
C84D:  4C 42 C1   JMP  $C142     sonst Fenster löschen
C850:  09 80      ORA  # $80     Setze Bit 7, da es eine Farbe
C852:  D0 86      BNE  $C7DA     sein muß und springe in Auswertung

*****************************    Cursor rechts im Fenster

C854:  20 ED CB   JSR  $CBED     Cursor um eine Stelle nach rechts
C857:  B0 04      BCS  $C85D     Neue Zeile begonnen
C859:  60         RTS            Rücksprung aus dem Unterprogramm

*****************************    Cursor Down

C85A:  20 63 C3   JSR  $C363     Zeilenvorschub ausführen
C85D:  20 74 CB   JSR  $CB74     Zeilenüberlaufbit testen
C860:  B0 03      BCS  $C865     Zeile hatte Überlänge
C862:  38         SEC            Setze Carry und rotiere
C863:  66 E8      ROR  # $E8     es in die Starteingabezeile
C865:  18         CLC            Lösche Carry für OK
C866:  60         RTS            Rücksprung aus dem Unterprogramm


*****************************    Cursor Up

C867:  A6 E5      LDX  * $E5     Obere Fenstergrenze in X-Reg laden
C869:  E4 EB      CPX  * $EB     vergleiche m. aktueller Cursor Zeile
C86B:  B0 F9      BCS  $C866     ist kleiner oder gleich
C86D:  20 5D C8   JSR  $C85D     Zeilenstatus setzen
C870:  C6 EB      DEC  * $EB     aktuelle Cur. Zeile um 1 vermindern
C872:  4C 5C C1   JMP  $C15C     Startadresse akt. Zeile ermitteln

*****************************    Cursor links im Fenster

C875:  20 00 CC   JSR  $CC00     Cursor links
C878:  B0 EC      BCS  $C866     Cursor wurde nicht bewegt
C87A:  D0 E9      BNE  $C865     Cursor bewegt, keine neue Zeile
C87C:  E6 EB      INC  * $EB     aktuelle Cursor Zeile um 1 erhöhen
```

```
C87E:  D0 ED     BNE  $C86D     unbedingter Sprung

******************************  2. Zeichensatz

C880:  24 D7     BIT  * $D7     Teste 40/80-Zeichen-Modus
C882:  30 07     BMI  $C88B     Springe bei 80-Zeichen-Modus
C884:  AD 2C 0A  LDA  $0A2C     Hole CHARROM-Basisadresse
C887:  09 02     ORA  # $02     Setze Bits 0 und 1
C889:  D0 10     BNE  $C89B     Springe unbedingt
C88B:  A5 F1     LDA  * $F1     Farbcode für Zeichenausgabe in Akku
C88D:  09 80     ORA  # $80     Alternate-Zeichensatz auswählen
C88F:  85 F1     STA  * $F1     Farbcode für Zeichenausgabe sichern
C891:  60        RTS            Rücksprung aus dem Unterprogramm

******************************  <Shift> <Commodore>

C892:  24 D7     BIT  * $D7     Teste 40/80-Zeichen-Modus
C894:  30 09     BMI  $C89F     Bei 80-Zeichen-Modus springen
C896:  AD 2C 0A  LDA  $0A2C     Hole Basisadresse CHARROM
C899:  29 FD     AND  # $FD     Lösche Bits 0 und 1
C89B:  8D 2C 0A  STA  $0A2C     als neue Basisadresse abspeichern
C89E:  60        RTS            Rücksprung aus dem Unterprogramm

******************************  <Shift> <Commodore> 80-Zeichen

C89F:  A5 F1     LDA  * $F1     Farbcode für Zeichenausgabe in Akku
C8A1:  29 7F     AND  # $7F     Bit 7 löschen, erster Zeichensatz
C8A3:  85 F1     STA  * $F1     Farbcode für Zeichenausgabe sichern
C8A5:  60        RTS            Rücksprung aus dem Unterprogramm

******************************  <Shift> <Commodore> sperren/freig.

C8A6:  A9 80     LDA  # $80     Bit 7 setzen zum Sperren und mit
C8A8:  05 F7     ORA  * $F7     Flagregister verORen
C8AA:  30 04     BMI  $C8B0     Unbedingter Sprung
C8AC:  A9 7F     LDA  # $7F     Bit 7 löschen, um Tasten
C8AE:  25 F7     AND  * $F7     freizugeben
C8B0:  85 F7     STA  * $F7     und abspeichern
C8B2:  60        RTS            Rücksprung aus dem Unterprogramm

******************************  Teste auf <Home>-<Home>-Kombination

C8B3:  A5 F0     LDA  * $F0     Hole zuletzt ausgegebenes Zeichen
C8B5:  C9 13     CMP  # $13     War es HOME?
C8B7:  D0 03     BNE  $C8BC     Wenn nicht, dann Ende der Routine
C8B9:  20 24 CA  JSR  $CA24     sonst löse Fenster auf
C8BC:  4C 50 C1  JMP  $C150     Springe nach Cursor-Home
```

```
*****************************     Revers-Modus setzen/löschen

C8BF:  A9 00     LDA  # $00      Akku mit Null laden, RVS löschen
C8C1:  2C        .Byte  $2C      Skip nach $C8C4
C8C2:  A9 80     LDA  # $80      Bit 7 setzen, RVS-Modus einschalten
C8C4:  85 F3     STA  * $F3      und Flag abspeichern
C8C6:  60        RTS             Rücksprung aus dem Unterprogramm

*****************************     Unterstreichen einschalten

C8C7:  A5 F1     LDA  * $F1      Farbcode für Zeichenausgabe in Akku
C8C9:  09 20     ORA  # $20      Bit 5 setzen für Unterstreichen ein
C8CB:  85 F1     STA  * $F1      Farbcode für Zeichenausgabe sichern
C8CD:  60        RTS             Rücksprung aus dem Unterprogramm

*****************************     Unterstreichen ausschalten

C8CE:  A5 F1     LDA  * $F1      Farbcode für Zeichenausgabe in Akku
C8D0:  29 DF     AND  # $DF      Bit 5 löschen, Unterstreichen aus
C8D2:  85 F1     STA  * $F1      Farbcode für Zeichenausgabe sichern
C8D4:  60        RTS             Rücksprung aus dem Unterprogramm

*****************************     Blink-Modus setzen

C8D5:  A5 F1     LDA  * $F1      Farbcode für Zeichenausgabe in Akku
C8D7:  09 10     ORA  # $10      Bit 4 setzen für Flash ein
C8D9:  85 F1     STA  * $F1      Farbcode für Zeichenausgabe sichern
C8DB:  60        RTS             Rücksprung aus dem Unterprogramm

*****************************     Blink-Modus ausschalten

C8DC:  A5 F1     LDA  * $F1      Farbcode für Zeichenausgabe in Akku
C8DE:  29 EF     AND  # $EF      Bit 4 löschen, kein Flash
C8E0:  85 F1     STA  * $F1      Farbcode für Zeichenausgabe sichern
C8E2:  60        RTS             Rücksprung aus dem Unterprogramm

*****************************     Insert ausführen

C8E3:  20 1E CC  JSR  $CC1E      Kopiere Cursor-Koordinaten
C8E6:  20 C3 CB  JSR  $CBC3      Suche Ende der Eingabezeile
C8E9:  E4 DF     CPX  * $DF      vergleiche Zeile mit Cursorzeile
C8EB:  D0 02     BNE  $C8EF      Hat sich geändert, dann Sprung
C8ED:  C4 DE     CPY  * $DE      vergleiche Spalte mit akt. Spalte
C8EF:  90 21     BCC  $C912      ist kleiner geworden
C8F1:  20 3E C3  JSR  $C33E      Cursor ans Zeilenende
C8F4:  B0 22     BCS  $C918      Es kann nicht gescrollt werden
C8F6:  20 00 CC  JSR  $CC00      Cursor um eins nach links
C8F9:  20 58 CB  JSR  $CB58      Hole Zeichen und Farbe Cursorpos.
```

```
C8FC:  20 ED CB   JSR  $CBED      Cursor wieder um eine Stelle rechts
C8FF:  20 32 CC   JSR  $CC32      Zeichen ausgeben
C902:  20 00 CC   JSR  $CC00      Cursor um eine Stelle nach links
C905:  A6 EB      LDX  * $EB      Hole aktuelle Cursor Zeile in X-Reg
C907:  E4 DF      CPX  * $DF      vergleiche mit Anfangscursorzeile
C909:  D0 EB      BNE  $C8F6      nächstes Zeichen kopieren
C90B:  C4 DE      CPY  * $DE      vergleiche Spalte mit Anfangsspalte
C90D:  D0 E7      BNE  $C8F6      wenn noch nicht erreicht, weiter
C90F:  20 27 CC   JSR  $CC27      Space an aktuelle Cursorposition
C912:  E6 F5      INC  * $F5      Erhöhe den Zähler für Insert
C914:  D0 02      BNE  $C918      Wenn ungleich null, dann Sprung
C916:  C6 F5      DEC  * $F5      sonst Insert wieder zurücksetzen
C918:  4C 32 C9   JMP  $C932      alte Cursorpos. zurücksetzen

*****************************     Zeichen links von Cursor löschen

C91B:  20 75 C8   JSR  $C875      Cursor links mit Bitbeeinflussung
C91E:  20 1E CC   JSR  $CC1E      Kopiere die Cursor-Koordinate
C921:  B0 0F      BCS  $C932      Cursor links nicht möglich
C923:  C4 E7      CPY  * $E7      vergleiche mit rechter Fenstergrenze
C925:  90 16      BCC  $C93D      Grenze noch nicht erreicht
C927:  A6 EB      LDX  * $EB      Hole aktuelle Cursor Zeile in X-Reg
C929:  E8         INX             erhöhe die Zeile um 1
C92A:  20 76 CB   JSR  $CB76      Teste Überlauf-Bit
C92D:  B0 0E      BCS  $C93D      Es gibt Folgezeile,
C92F:  20 27 CC   JSR  $CC27      sonst <Space> an akt. Position

******************************    Alte Cursor-Adresse wieder setzen

C932:  A5 DE      LDA  * $DE      Hole zwischengespeicherte Spalte
C934:  85 EC      STA  * $EC      sichern der aktuellen Cursor Spalte
C936:  A5 DF      LDA  * $DF      Hole zwischengespeicherte Zeile
C938:  85 EB      STA  * $EB      aktuelle Cursor Zeile zurückschreib.
C93A:  4C 5C C1   JMP  $C15C      Ermittle Startadresse der Zeile

******************************    Zeichen unter Cursor löschen

C93D:  20 ED CB   JSR  $CBED      Cursor um eins nach rechts
C940:  20 58 CB   JSR  $CB58      Hole Zeichen und Farbe an Cursor
C943:  20 00 CC   JSR  $CC00      Cursor wieder um 1 nach links
C946:  20 32 CC   JSR  $CC32      Zeichen an Cursorpos
C949:  20 ED CB   JSR  $CBED      Cursor wieder nach rechts
C94C:  4C 23 C9   JMP  $C923      Zeile nach Cursor verschieben

******************************    Tabulator anspringen

C94F:  A4 EC      LDY  * $EC      Hole aktuelle Cursor Spalte in Y-Reg
C951:  C8         INY             Erhöhe den Spaltenzeiger
```

```
C952:  C4 E7     CPY  * $E7      vergleiche mit rechter Fenstergrenze
C954:  B0 06     BCS  $C95C      Keine Tabs mehr möglich
C956:  20 6C C9  JSR  $C96C      Hole nächste Tab-Position
C959:  F0 F6     BEQ  $C951      Cursor steht auf Tab-Pos., nochmal
C95B:  2C        .Byte  $2C      Skip nach $C95E
C95C:  A4 E7     LDY  * $E7      Rechte Fenstergrenze nach Y
C95E:  84 EC     STY  * $EC      sichern der aktuellen Cursor Spalte
C960:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   Tabulator setzen/löschen

C961:  A4 EC     LDY  * $EC      Hole aktuelle Cursor Spalte in Y-Reg
C963:  20 6C C9  JSR  $C96C      Hole Tabulatorbyte
C966:  45 DA     EOR  * $DA      Umkehren des Tabulatorbits
C968:  9D 54 03  STA  $0354,X    und wieder abspeichern
C96B:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   Tabulatorposition ermitteln

C96C:  98        TYA             Spalte in Akkumulator kopieren
C96D:  29 07     AND  # $07      Bits 4-7 ausmaskieren=A MOD 7
C96F:  AA        TAX             Und nach X-Register als Zeiger
C970:  BD 6C CE  LDA  $CE6C,X    Hole Zweierpotenz
C973:  85 DA     STA  * $DA      und in $DA zwischenspeichern
C975:  98        TYA             Spalte wieder in Akku
C976:  4A        LSR  A          Akku wird dreimal
C977:  4A        LSR  A          nach rechts geshifted, was
C978:  4A        LSR  A          INT (A/8) gleichkommt
C979:  AA        TAX             wieder nach X-Register als Zeiger
C97A:  BD 54 03  LDA  $0354,X    Tabulatorbyte holen
C97D:  24 DA     BIT  * $DA      Teste, ob 8. Tab gesetzt ist
C97F:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   Löschen der Tabs (bzw. Reset)

C980:  A9 00     LDA  # $00      Lade Akku mit Null, um zu löschen
C982:  2C        .Byte  $2C      Skip nach $C985
C983:  A9 80     LDA  # $80      Jede 8. Stelle ist Tabulator
C985:  A2 09     LDX  # $09      Es werden alle 10 Tab-Bytes
C987:  9D 54 03  STA  $0354,X    mit dem Wert beschrieben
C98A:  CA        DEX             Erniedrige Zähler und
C98B:  10 FA     BPL  $C987      springe, wenn noch nicht fertig
C98D:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   CHR$(7) Bell-Ton erklingt

C98E:  24 F9     BIT  * $F9      Teste Beep-Flag
C990:  30 FB     BMI  $C98D      Kein Ton, bitte!
```

```
C992:  A9 15     LDA  # $15      Die Lautstärke des SIDs wird
C994:  8D 18 D4  STA  $D418      auf 15 (maximal) gesetzt!
C997:  A0 09     LDY  # $09      Attack/Decay-Konstante
C999:  A2 00     LDX  # $00      Sustain/Release-Konstante
C99B:  8C 05 D4  STY  $D405      in die entsprechenden Register
C99E:  8E 06 D4  STX  $D406      ablegen (für Stimme 1)
C9A1:  A9 30     LDA  # $30      Hi-Byte der Frequenz
C9A3:  8D 01 D4  STA  $D401      für Stimme 1 definieren
C9A6:  A9 20     LDA  # $20      Sägezahnschwingung auswählen
C9A8:  8D 04 D4  STA  $D404      und SID mitteilen
C9AB:  A9 21     LDA  # $21      Durch Setzen des Bits 0 wird
C9AD:  8D 04 D4  STA  $D404      der Ton angeschlagen
C9B0:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   <LF> - Cursor-Spalte bleibt

C9B1:  A5 EC     LDA  * $EC      Hole aktuelle Cursor Spalte in Akku
C9B3:  48        PHA             Rette akt. Spalte auf Akku
C9B4:  20 C3 CB  JSR  $CBC3      Suche Ende der Zeile
C9B7:  20 63 C3  JSR  $C363      Zeilenvorschub ausführen
C9BA:  68        PLA             Hole aktuelle Spalte wieder
C9BB:  85 EC     STA  * $EC      sichern der aktuellen Cursor Spalte
C9BD:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   ESC-Sequenzen ausführen

C9BE:  6C 38 03  JMP  ($0338)    Vektor Zeichenausgabe mit Esc
C9C1:  C9 1B     CMP  # $1B      ist Zeichen <ESC>?
C9C3:  D0 05     BNE  $C9CA      Springe, wenn anderes Zeichen
C9C5:  46 EF     LSR  * $EF      Aktuelles Zeichen durch 2
C9C7:  4C 7D C7  JMP  $C77D      Alle Sonderfunktionen ausschalten
C9CA:  29 7F     AND  # $7F      Bit 7 ausblenden, keine RVS Zeichen
C9CC:  38        SEC             Setze Carry für Subtraktion
C9CD:  E9 40     SBC  # $40      Subtrahiere 64 von ASCII-Wert
C9CF:  C9 1B     CMP  # $1B      Vergleiche mit 27
C9D1:  B0 0A     BCS  $C9DD      Rückkehr, wenn Zeichen größer als Z
C9D3:  0A        ASL  A          Akku mal 2, da 16-Bit-Wert geholt
C9D4:  AA        TAX             wird und nach X als Zeiger
C9D5:  BD DF C9  LDA  $C9DF,X    Hole Hi-Byte der Ausführungsroutine
C9D8:  48        PHA             Rette auf Stack
C9D9:  BD DE C9  LDA  $C9DE,X    Hole Lo-Byte der Routine
C9DC:  48        PHA             auf Stack. Springe Routine durch
C9DD:  60        RTS             RTS an. Adresse ist auf Stack

******************************   Adressen der ESC-Routinen

C9DE:  9E CA     $CA9E           <ESC> @ - Lösche Cursor bis Ende
C9E0:  EC CA     $CAEC           <ESC> A - Auto-Insert ein
```

```
C9E2:  15 CA      $CA15        <ESC> B - Setze unteren Fensterrand
C9E4:  E9 CA      $CAE9        <ESC> C - Auto-Insert aus
C9E6:  51 CA      $CA51        <ESC> D - Lfd. Zeile löschen
C9E8:  0A CB      $CB0A        <ESC> E - Cursor blinken aus
C9EA:  20 CB      $CB20        <ESC> F - Cursor blinken ein
C9EC:  36 CB      $CB36        <ESC> G - Signalton  zulassen
C9EE:  39 CB      $CB39        <ESC> H - Signalton verhindern
C9F0:  3C CA      $CA3C        <ESC> I - Zeile einfügen
C9F2:  B0 CB      $CBB0        <ESC> J - Cursor an Anfang Zeile
C9F4:  51 CB      $CB51        <ESC> K - Cursor an Ende Zeile
C9F6:  E1 CA      $CAE1        <ESC> L - Scrollen zulassen
C9F8:  E4 CA      $CAE4        <ESC> M - Scrollen verhindern
C9FA:  47 CB      $CB47        <ESC> N - Revers aus (80er)
C9FC:  7C C7      $C77C        <ESC> O - Insert, Quote, RVS aus
C9FE:  8A CA      $CA8A        <ESC> P - Cursor bis Zeilenbeginn
CA00:  75 CA      $CA75        <ESC> Q - Cursor bis Zeilenende clr
CA02:  3E CB      $CB3E        <ESC> R - Reverser Bildschirm (80)
CA04:  F1 CA      $CAF1        <ESC> S - Block-Cursor (80er)
CA06:  13 CA      $CA13        <ESC> T - Oberen Fensterrand setzen
CA08:  FD CA      $CAFD        <ESC> U - Underline Cursor (80er)
CA0A:  BB CA      $CABB        <ESC> V - Aufwärts scrollen
CA0C:  C9 CA      $CAC9        <ESC> W - Abwärts scrollen
CA0E:  2B CD      $CD2B        <ESC> X - Umschalten 40/80-Zeichen
CA10:  82 C9      $C982        <ESC> Y - Alle TABs normal setzen
CA12:  7F C9      $C97F        <ESC> Z - Alle TABs löschen

*******************************  Definition von Fenstergrenzen

CA14:  18         CLC          Cursorposition ist links/oben
CA15:  24         .Byte  $24   Skip nach $CA17
CA16:  38         SEC          Cursorposition ist rechts/unten
CA17:  A6 EC      LDX  * $EC   Hole aktuelle Cursor-Spalte in X-Reg
CA19:  A5 EB      LDA  * $EB   Hole aktuelle Cursor-Zeile in Akku
CA1B:  90 11      BCC  $CA2E   Bei gelöschtem Carry: links/oben!
CA1D:  85 E4      STA  * $E4   Untere Fenstergrenze definieren
CA1F:  86 E7      STX  * $E7   sowie rechte Grenze des Fensters
CA21:  4C 32 CA   JMP  $CA32   Rest der Routine ausführen

*******************************  Bildschirm als Fenster definieren

CA24:  A5 ED      LDA  * $ED   Hole Maximalzeilenzahl in Akku
CA26:  A6 EE      LDX  * $EE   Hole Maximalspaltenzahl nach X
CA28:  20 1D CA   JSR  $CA1D   Als rechts/unten definieren
CA2B:  A9 00      LDA  # $00   Links oben mit 0/0
CA2D:  AA         TAX          belegen und
CA2E:  85 E5      STA  * $E5   auch als linke und
CA30:  86 E6      STX  * $E6   obere Grenze definieren
CA32:  A9 00      LDA  # $00   Akku wird mit Null geladen und
```

```
CA34:  A2 04     LDX  # $04      das X-Register mit 4, um die
CA36:  9D 5D 03  STA  $035D,X    Zeilenüberlaufbits zu löschen
CA39:  CA        DEX             Erniedrige Zähler und springe,
CA3A:  D0 FA     BNE  $CA36      wenn noch nicht alle Bits gelöscht
CA3C:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   Zeile einfügen

CA3D:  20 7C C3  JSR  $C37C      Restbildschirm nach X verschieben
CA40:  20 56 C1  JSR  $C156      Cursor links und Startad. ermitteln
CA43:  E8        INX             erhöhe die Zeile
CA44:  20 76 CB  JSR  $CB76      Zeilenüberlaufbit testen
CA47:  08        PHP             Rette das Carry
CA48:  20 81 CB  JSR  $CB81      Zeilenüberlaufbit setzen/löschen
CA4B:  28        PLP             Hole Carry wieder von Stack
CA4C:  B0 03     BCS  $CA51      Cursorzeile ist Startzeile
CA4E:  38        SEC             sonst markiere akt. Zeile
CA4F:  66 E8     ROR  # $E8      als Folgezeile
CA51:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   Laufende Zeile löschen

CA52:  20 B5 CB  JSR  $CBB5      Zeilenanfangsadresse setzen
CA55:  A5 E5     LDA  * $E5      Obere Fenstergrenze in Akku laden
CA57:  48        PHA             Rette Fenstergrenze auf Stack
CA58:  A5 EB     LDA  * $EB      Hole aktuelle Cursor-Zeile in Akku
CA5A:  85 E5     STA  * $E5      als obere Fenstergrenze definieren
CA5C:  A5 F8     LDA  * $F8      Scrollflag auf
CA5E:  48        PHA             Stack sichern
CA5F:  A9 80     LDA  # $80      Scrollen erst einmal nicht
CA61:  85 F8     STA  * $F8      ermöglichen
CA63:  20 B8 C3  JSR  $C3B8      Aufwärts scrollen
CA66:  68        PLA             Hole Scrollflag wieder zurück
CA67:  85 F8     STA  * $F8      und auch wieder rekonstruieren
CA69:  A5 E5     LDA  * $E5      Obere Fenstergrenze in Akku laden
CA6B:  85 EB     STA  * $EB      aktuelle Cursor Zeile zurückschreib.
CA6D:  68        PLA             Hole obere Fenstergrenze
CA6E:  85 E5     STA  * $E5      und wieder zurückspeichern
CA70:  38        SEC             Setze Carry, um in $E8 hineinzu-
CA71:  66 E8     ROR  # $E8      schieben; markiere als Folgezeile
CA73:  4C 56 C1  JMP  $C156      Cursor linke Fenstergrenze

******************************   Cursor bis Zeilenende löschen

CA76:  20 1E CC  JSR  $CC1E      Rette Cursorkoordinaten
CA79:  20 AA C4  JSR  $C4AA      Lösche akt. Zeile ab Cursor
CA7C:  E6 EB     INC  * $EB      aktuelle Cursor Zeile um 1 erhöhen
CA7E:  20 5C C1  JSR  $C15C      Ermittle Startadresse Zeile
```

```
CA81:  A4 E6     LDY  * $E6     Linke Fenstergrenze in Y-Reg laden
CA83:  20 74 CB  JSR  $CB74     Zeilenüberlaufbit testen
CA86:  B0 F1     BCS  $CA79     Folgezeile auch noch löschen
CA88:  4C 32 C9  JMP  $C932     Alte Cursoradresse zurücksetzen

******************************  Zeilenanfang bis Cursor löschen

CA8B:  20 1E CC  JSR  $CC1E     Rette Cursorkoordinaten
CA8E:  20 27 CC  JSR  $CC27     Space an aktuelle Cursorposition
CA91:  C4 E6     CPY  * $E6     vergleiche mit linker Fenstergrenze
CA93:  D0 05     BNE  $CA9A     Noch nicht erreicht
CA95:  20 74 CB  JSR  $CB74     Teste Zeilenüberlaufbit
CA98:  90 EE     BCC  $CA88     Kein Überlauf - dann Ende
CA9A:  20 00 CC  JSR  $CC00     Sonst Cusor links
CA9D:  90 EF     BCC  $CA8E     Wenn bewegt, dann lösche die Zeile

******************************  Lösche Cursorp. bis Bildschirmende

CA9F:  20 1E CC  JSR  $CC1E     Rette Cursorkoordinaten
CAA2:  20 AA C4  JSR  $C4AA     Lösche Zeile
CAA5:  E6 EB     INC  * $EB     aktuelle Cursor Zeile um 1 erhöhen
CAA7:  20 5C C1  JSR  $C15C     Ermittle Startadresse Cursorzeile
CAAA:  A4 E6     LDY  * $E6     Linke Fenstergrenze in Y-Reg laden
CAAC:  20 74 CB  JSR  $CB74     Teste Zeilenüberlaufbit
CAAF:  B0 F1     BCS  $CAA2     Zeile noch nicht zu Ende
CAB1:  A5 EB     LDA  * $EB     Hole aktuelle Cursor Zeile in Akku
CAB3:  C5 E4     CMP  * $E4     vergleiche mit unterer Fenstergrenze
CAB5:  90 EB     BCC  $CAA2     Untere Grenze noch nicht erreicht
CAB7:  F0 E9     BEQ  $CAA2     Untere Grenze gerade erreicht
CAB9:  4C 32 C9  JMP  $C932     Alte Cusoradresse rücksetzen

******************************  Aufwärts scrollen

CABC:  20 1E CC  JSR  $CC1E     Rette Cursorkoordinaten
CABF:  8A        TXA            Zeile in Akku und
CAC0:  48        PHA            dann auf Stack sichern
CAC1:  20 A6 C3  JSR  $C3A6     Aufwärts scrollen ausführen
CAC4:  68        PLA            Hole Zeile wieder von Stack
CAC5:  85 DF     STA  * $DF     und sichern
CAC7:  4C 32 C9  JMP  $C932     Alte Cursorkoordinaten zurück

******************************  Abwärts scrollen

CACA:  20 1E CC  JSR  $CC1E     Rette Cursorkoordinaten
CACD:  20 74 CB  JSR  $CB74     Teste Zeilenüberlaufbit
CAD0:  B0 03     BCS  $CAD5     Zeile ist keine Überlaufseile
CAD2:  38        SEC            Markiere, daß Eingabezeile nicht
CAD3:  66 E8     ROR  # $E8     Startzeile ist
```

```
CAD5:  A5 E5     LDA  * $E5    Obere Fenstergrenze in Akku laden
CAD7:  85 EB     STA  * $EB    aktuelle Cursor Zeile zurückschreib.
CAD9:  20 7C C3  JSR  $C37C    Abwärts scrollen
CADC:  20 85 CB  JSR  $CB85    Lösche Zeilenüberlaufbit
CADF:  4C 32 C9  JMP  $C932    Alte Cursorkoordinaten zurück

******************************  Scrollen zulassen/verhindern

CAE2:  A9 00     LDA  # $00    Scrollen zulassen
CAE4:  2C        .Byte  $2C    Skip nach $CAaE7
CAE5:  A9 80     LDA  # $80    Scrollen unterbinden
CAE7:  85 F8     STA  * $F8    Scrollflag abspeichern
CAE9:  60        RTS           Rücksprung aus dem Unterprogramm

******************************  Flag für Auto-Insert setzen/löschen

CAEA:  A9 00     LDA  # $00    Auto-Insert-Flag löschen
CAEC:  2C        .Byte  $2C    Skip nach $CAEF
CAED:  A9 80     LDA  # $80    Auto-Insert-Flag setzen
CAEF:  85 F6     STA  * $F6    und Flag sichern
CAF1:  60        RTS           Rücksprung aus dem Unterprogramm

******************************  Blockcursor einschalten

CAF2:  24 D7     BIT  * $D7    Teste 40/80-Zeichen-Modus
CAF4:  10 40     BPL  $CB36    Bei 40-Zeichen -> Ende
CAF6:  AD 2B 0A  LDA  $0A2B    Hole VDC-Cursor-Modus
CAF9:  29 E0     AND  # $E0    Bits 0-4 (Start-Scan) ausmaskieren
CAFB:  4C 14 CB  JMP  $CB14    Abspeichern und VIC-Cursor aus

******************************  Underline-Cursor einschalten

CAFE:  24 D7     BIT  * $D7    Teste 40/80-Zeichen-Flag
CB00:  10 34     BPL  $CB36    Wenn 40 Zeichen, Ende
CB02:  AD 2B 0A  LDA  $0A2B    Hole VDC-Cursor-Modus
CB05:  29 E0     AND  # $E0    Start-Scan ausmaskieren
CB07:  09 07     ORA  # $07    Start-Scan-Line ist 7
CB09:  D0 09     BNE  $CB14    unbedingter Sprung zum Setzen

******************************  Cursor: Blinken aus

CB0B:  24 D7     BIT  * $D7    Teste 40/80-Zeichen Modus
CB0D:  10 0B     BPL  $CB1A    Wenn 40 Zeichen, dann Sprung
CB0F:  AD 2B 0A  LDA  $0A2B    Hole VDC-Cursor-Modus
CB12:  29 1F     AND  # $1F    Blinken ausmaskieren
CB14:  8D 2B 0A  STA  $0A2B    und wieder abspeichern
CB17:  4C 91 CD  JMP  $CD91    Setze Modus und VIC aus
```

```
*****************************   für 40 Zeichen

CB1A:  AD 26 0A  LDA  $0A26     Hole VIC-Cursor-Modus
CB1D:  09 40     ORA  # $40     Setze Bit 6 für "starr"
CB1F:  D0 12     BNE  $CB33     springe unbedingt zum Speichern

*****************************   Cursor: Blinken ein

CB21:  24 D7     BIT  * $D7     Teste 40/80-Zeichen-Modus
CB23:  10 09     BPL  $CB2E     Springe, wenn 40 Zeichen
CB25:  AD 2B 0A  LDA  $0A2B     Hole VDC-Cursor-Modus
CB28:  29 1F     AND  # $1F     Blinken ausmaskieren
CB2A:  09 60     ORA  # $60     und Blinkperiode definieren
CB2C:  D0 E6     BNE  $CB14     springe unbedingt zum Setzen

*****************************   für 40 Zeichen

CB2E:  AD 26 0A  LDA  $0A26     Hole VIC-Cursor-Modus
CB31:  29 BF     AND  # $BF     Bit 6 (starr) ausmaskieren
CB33:  8D 26 0A  STA  $0A26     und wieder abspeichern
CB36:  60        RTS            Rücksprung aus dem Unterprogramm

*****************************   Flag für Bell setzen/löschen

CB37:  A9 00     LDA  # $00     Bell ermöglichen
CB39:  2C        .Byte  $2C     Skip nach $CB3C
CB3A:  A9 80     LDA  # $80     Bell sperren
CB3C:  85 F9     STA  * $F9     und Flag abspeichern
CB3E:  60        RTS            Rücksprung aus dem Unterprogramm

*****************************   80-Zeichen-Monitor revers schalten

CB3F:  A2 18     LDX  # $18     Register 24 auswählen
CB41:  20 DA CD  JSR  $CDDA     und momentanen Inhalt holen
CB44:  09 40     ORA  # $40     Setze Revers-Flag
CB46:  D0 07     BNE  $CB4F     unbedingter Sprung nach $CB4F

*****************************   80-Zeichen-Monitor normal schalten

CB48:  A2 18     LDX  # $18     Register 24 auswählen
CB4A:  20 DA CD  JSR  $CDDA     und momentanen Inhalt holen
CB4D:  29 BF     AND  # $BF     Lösche das Revers-Flag
CB4F:  4C CC CD  JMP  $CDCC     und abspeichern

*****************************   Cursor ans Ende lfd. Zeile

CB52:  20 C3 CB  JSR  $CBC3     Ermittle Startadresse lfd. Zeile
```

```
CB55:  4C 3E C3   JMP  $C33E      Cursor ans Zeilenende

*****************************     Hole Zeichen und Farbe an Cursorpos

CB58:  A4 EC      LDY  * $EC      Hole aktuelle Cursor Spalte in Y-Reg
CB5A:  24 D7      BIT  * $D7      Teste 40/80-Zeichen-Modus
CB5C:  30 07      BMI  $CB65      Springe, wenn 80-Zeichen-Modus
CB5E:  B1 E2      LDA  ($E2),Y    Hole Farbe an Cursorposition
CB60:  85 F2      STA  * $F2      Und abspeichern
CB62:  B1 E0      LDA  ($E0),Y    Hole Zeichen an Cursorposition
CB64:  60         RTS             Rücksprung aus dem Unterprogramm

*****************************     Hole Zeichen und Farbe unter Cursor

CB65:  20 F9 CD   JSR  $CDF9      Setzen der Update-Adresse auf ARA
CB68:  20 D8 CD   JSR  $CDD8      Hole aktuelles Attribut
CB6B:  85 F2      STA  * $F2      Speichere Attribut
CB6D:  20 E6 CD   JSR  $CDE6      Setzen der Update-Adresse auf Video
CB70:  20 D8 CD   JSR  $CDD8      Hole Zeichen aus Video-RAM
CB73:  60         RTS             Rücksprung aus dem Unterprogramm

*****************************     Routinen zum Testen des Zeilen-
                                  überlaufbit
CB74:  A6 EB      LDX  * $EB      Hole aktuelle Cursor Zeile in X-Reg
CB76:  20 9F CB   JSR  $CB9F      Zweierpotenz und Rest ermitteln
CB79:  3D 5E 03   AND  $035E,X    Zeilenüberlaufbit löschen
CB7C:  C9 01      CMP  # $01      Ist keine Zeile gesetzt in dem
CB7E:  4C 90 CB   JMP  $CB90      Block? Springe zum Ende der Routine
CB81:  A6 EB      LDX  * $EB      Hole aktuelle Cursor Zeile in X-Reg
CB83:  B0 0E      BCS  $CB93      Springe, wenn Flag gesetzt
CB85:  20 9F CB   JSR  $CB9F      Zweierpotenz und Rest ermitteln
CB88:  49 FF      EOR  # $FF      Einerkomplement von Akku
CB8A:  3D 5E 03   AND  $035E,X    und mit Zeilenüberlauftabelle
CB8D:  9D 5E 03   STA  $035E,X    verknüpfen und wieder abspeichern
CB90:  A6 DA      LDX  * $DA      Hole X aus Zwischenspeicher
CB92:  60         RTS             Rücksprung aus dem Unterprogramm

*****************************     Setzen der Zeilenüberlaufbits

CB93:  24 F8      BIT  * $F8      Teste Scroll Bit
CB95:  70 DF      BVS  $CB76      Springe bei gesetztem 6. Bit
CB97:  20 9F CB   JSR  $CB9F      Ermittle Zweierpotenz und Rest
CB9A:  1D 5E 03   ORA  $035E,X    Setzen des Zeilenüberlaufbits
CB9D:  D0 EE      BNE  $CB8D      und updaten.

*****************************     Routine ermittelt 2^(X AND 7) sowie
                                  INT (X/8). Übergabe in X-Register
```

```
CB9F:  86 DA     STX  * $DA      Zwischenspeichern des Akkus
CBA1:  8A        TXA             X-Register in Akku
CBA2:  29 07     AND  # $07      Ausmaskieren der Bits 3-7=X MOD 8
CBA4:  AA        TAX             Akku wieder in X-Register
CBA5:  BD 6C CE  LDA  $CE6C,X    Hole entsprechende Zweierpotenz
CBA8:  48        PHA             Rette Akku auf Stack
CBA9:  A5 DA     LDA  * $DA      Hole Originalwert wieder
CBAB:  4A        LSR  A          Dieser Originalwert wird dreimal
CBAC:  4A        LSR  A          durch den Faktor 2 dividiert,
CBAD:  4A        LSR  A          das ergiebt INT(X/8)
CBAE:  AA        TAX             Ergebnis - X-Register
CBAF:  68        PLA             Hole Zweierpotenz von Stack
CBB0:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   Löschen der Überlaufverkettung

CBB1:  A4 E6     LDY  * $E6      Linke Fenstergrenze in Y-Reg laden
CBB3:  84 EC     STY  * $EC      sichern der aktuellen Cursor Spalte
CBB5:  20 74 CB  JSR  $CB74      Lösche Zeilenüberlaufbit akt. Zeile
CBB8:  90 06     BCC  $CBC0      Carry ist gelöscht, wenn alle Bits 0
CBBA:  C6 EB     DEC  * $EB      aktuelle Cur. Zeile um 1 vermindern
CBBC:  10 F7     BPL  $CBB5      Wenn nicht erste Zeile, dann Sprung
CBBE:  E6 EB     INC  * $EB      aktuelle Cursor Zeile um 1 erhöhen
CBC0:  4C 5C C1  JMP  $C15C      Ermittle Startadresse akt. Zeile

******************************   Suche Ende der Eingabezeile

CBC3:  E6 EB     INC  * $EB      Aktuelle Cursor Zeile um 1 erhöhen
CBC5:  20 74 CB  JSR  $CB74      Lösche Zeilenüberlaufbit
CBC8:  B0 F9     BCS  $CBC3      Wenn nicht letzte Zeile => Sprung
CBCA:  C6 EB     DEC  * $EB      aktuelle Cur. Zeile um 1 vermindern
CBCC:  20 5C C1  JSR  $C15C      Ermittle Startadresse akt. Zeile
CBCF:  A4 E7     LDY  * $E7      Rechte Fenstergrenze in Y-Reg laden
CBD1:  84 EC     STY  * $EC      Sichern der aktuellen Cursor Spalte
CBD3:  20 58 CB  JSR  $CB58      Hole Zeichen und Farbe Cursorpos
CBD6:  A6 EB     LDX  * $EB      Hole aktuelle Cursor Zeile in X-Reg
CBD8:  C9 20     CMP  # $20      Ist Zeichen <Space>?
CBDA:  D0 0E     BNE  $CBEA      Nein, dann Sprung
CBDC:  C4 E6     CPY  * $E6      Vergleiche mit linker Fenstergrenze
CBDE:  D0 05     BNE  $CBE5      noch nicht erreicht
CBE0:  20 74 CB  JSR  $CB74      Lösche Zeilenüberlaufbit
CBE3:  90 05     BCC  $CBEA      Es ist noch eine Zeile frei
CBE5:  20 00 CC  JSR  $CC00      Cursor um eine Stelle nach links
CBE8:  90 E9     BCC  $CBD3      Cursor konnte bewegt werden
CBEA:  84 EA     STY  * $EA      Aktuelle Eingabezeile: Ende
CBEC:  60        RTS             Rücksprung aus dem Unterprogramm
```

```
******************************     Cursor um eine Stelle im Fenster
                                   nach rechts

CBED:  48        PHA               Rette Akku auf Stack
CBEE:  A4 EC     LDY  * $EC        Hole aktuelle Cursor Spalte in Y-Reg
CBF0:  C4 E7     CPY  * $E7        Vergleiche mit rechter Fenstergrenze
CBF2:  90 07     BCC  $CBFB        Rechte Grenze des Windows erreicht?
CBF4:  20 63 C3  JSR  $C363        Nein, dann erhöhe Cursor-Spalte
CBF7:  A4 E6     LDY  * $E6        Linke Fenstergrenze in Y-Reg laden
CBF9:  88        DEY               Einmal erniedrigen
CBFA:  38        SEC               Carry gesetzt bedeutet neue Zeile
CBFB:  C8        INY               Erhöhe Cursor-Spalte
CBFC:  84 EC     STY  * $EC        Sichern der aktuellen Cursor Spalte
CBFE:  68        PLA               Hole Akku wieder von Stack
CBFF:  60        RTS               Rücksprung aus dem Unterprogramm

******************************     Cursor um eine Stelle im Fenster
                                   nach links

CC00:  A4 EC     LDY  * $EC        Hole aktuelle Cursor Spalte in Y-Reg
CC02:  88        DEY               Erniedrige die Spalte um 1
CC03:  30 04     BMI  $CC09        Wenn negativ, Cursor in Spalte 0
CC05:  C4 E6     CPY  * $E6        Vergleiche mit linker Fenstergrenze
CC07:  B0 0F     BCS  $CC18        Linker Rand nicht erreicht, OK.
CC09:  A4 E5     LDY  * $E5        Obere Fenstergrenze in Y-Reg laden
CC0B:  C4 EB     CPY  * $EB        Vergleiche m. aktueller Cursor Zeile
CC0D:  B0 0E     BCS  $CC1D        Cursor ist in oberster Zeile, Ende
CC0F:  C6 EB     DEC  * $EB        Aktuelle Cur. Zeile um 1 vermindern
CC11:  48        PHA               Rette Akku auf Stack
CC12:  20 5C C1  JSR  $C15C        Ermittle Startadresse der Zeile
CC15:  68        PLA               Hole Akku wieder von Stack
CC16:  A4 E7     LDY  * $E7        Rechte Fenstergrenze in Y-Reg laden
CC18:  84 EC     STY  * $EC        Sichern der aktuellen Cursor Spalte
CC1A:  C4 E7     CPY  * $E7        Vergleiche mit rechter Fenstergrenze
CC1C:  18        CLC               Lösche Carry für Cursor bewegt
CC1D:  60        RTS               Rücksprung aus dem Unterprogramm

******************************     Kopiere Cursor (X/Y) nach $DE/$DF

CC1E:  A4 EC     LDY  * $EC        Hole aktuelle Cursor Spalte in Y-Reg
CC20:  84 DE     STY  * $DE        Kopiere nach $DE
CC22:  A6 EB     LDX  * $EB        Hole aktuelle Cursor Zeile in X-Reg
CC24:  86 DF     STX  * $DF        Kopiere nach $DF
CC26:  60        RTS               Rücksprung aus dem Unterprogramm

******************************     Space an akt. Cursor-Position

CC27:  A5 F1     LDA  * $F1        Farbcode für Zeichenausgabe in Akku
```

```
CC29:  29 8F     AND  # $8F    Bits 4-6 ausmaskieren (Attribute)
CC2B:  AA        TAX           Und nach X-Register
CC2C:  A9 20     LDA  # $20    Lade Akku mit Leerzeichen
CC2E:  2C        .Byte  $2C    Skip nach $CC31

****************************   Zeichen (Akku) an Cursor-Position

CC2F:  A6 F1     LDX  * $F1    Lade X-Register mit Farbe
CC31:  2C        .Byte  $2C    Skip nach $CC34
CC32:  A6 F2     LDX  * $F2    Farbcodespeicher für Insert/Delete
CC34:  A8        TAY           Akku nach Y-Register
CC35:  A9 02     LDA  # $02    Lege den Wert 2 in
CC37:  8D 28 0A  STA  $0A28    VIC-Cursor-Blink-Zähler
CC3A:  20 7C C1  JSR  $C17C    Passe Attribut-Adresse an
CC3D:  98        TYA           Und Y-Register wieder nach Akku
CC3E:  A4 EC     LDY  * $EC    Hole aktuelle Cursor Spalte in Y-Reg
CC40:  24 D7     BIT  * $D7    Teste 40/80-Zeichen-Modus
CC42:  30 06     BMI  $CC4A    Sprung bei 80-Zeichen-Modus
CC44:  91 E0     STA  ($E0),Y  Speichere Zeichen in 40-Zeichen
CC46:  8A        TXA           Video-RAM und X-Register (Farbe)
CC47:  91 E2     STA  ($E2),Y  im Farbspeicher ablegen.
CC49:  60        RTS           Rücksprung aus dem Unterprogramm

*****************************  Zeichen auf 80-Zeichen-Bildschirm
                               Akku: Zeichen, X: Farbe, Y: Spalte

CC4A:  48        PHA           Rette Akku auf Stack
CC4B:  8A        TXA           X-Register (Farbe) in Akku
CC4C:  48        PHA           und auch auf Stack sichern
CC4D:  20 F9 CD  JSR  $CDF9    Update-Register für Attribut setzen
CC50:  68        PLA           Hole Farbe von Stack in Akku
CC51:  20 CA CD  JSR  $CDCA    und ins Attribut-RAM speichern
CC54:  20 E6 CD  JSR  $CDE6    Update-Adresse für Video-RAM setzen
CC57:  68        PLA           und Zeichen von Stack holen
CC58:  4C CA CD  JMP  $CDCA    Speichere Zeichen im Video-RAM ab

****************************   Ermittle Zeichen/Zeile&Zeilen/Wind.

CC5B:  38        SEC           Setze Carry
CC5C:  A5 E4     LDA  * $E4    Untere Fenstergrenze in Akku laden
CC5E:  E5 E5     SBC  * $E5    minus oberer Grenze ergibt Zeilen
CC60:  A8        TAY           des Fensters nach Y-Register
CC61:  38        SEC           Setze das Carry wieder
CC62:  A5 E7     LDA  * $E7    Rechte Fenstergrenze in Akku laden
CC64:  E5 E6     SBC  * $E6    minus linke Fenstergrenze ergibt
CC66:  AA        TAX           Anzahl Zeichen/Zeile nach X-Reg
CC67:  A5 EE     LDA  * $EE    In Akku maximale Anzahl Spalten
```

```
CC69:  60          RTS          Rücksprung aus dem Unterprogramm

******************************  Hole oder setze Cursorposition

CC6A:  B0 29       BCS  $CC95   Wenn Carry gesetzt - dann hole Pos.
CC6C:  8A          TXA          Zeile nach Akku
CC6D:  65 E5       ADC  * $E5   Addiere obere Fenstergrenze
CC6F:  B0 14       BCS  $CC85   Wenn Übertrag, dann Ende (Fehler!)
CC71:  C5 E4       CMP  * $E4   vergleiche mit unterer Fenstergrenze
CC73:  F0 02       BEQ  $CC77   Wenn erreicht, dann OK!
CC75:  B0 0E       BCS  $CC85   Wenn Übertrag, dann Ende (Fehler!)
CC77:  48          PHA          Rette Zeile auf Stack
CC78:  18          CLC          Lösche Carry für Addition
CC79:  98          TYA          Hole Spalte in Akku
CC7A:  65 E6       ADC  * $E6   Und addiere linke Fenstergrenze
CC7C:  B0 06       BCS  $CC84   Wenn Übertrag - dann Ende (Fehler!)
CC7E:  C5 E7       CMP  * $E7   vergleiche mit rechter Fenstergrenze
CC80:  F0 04       BEQ  $CC86   Wenn gleich, dann OK
CC82:  90 02       BCC  $CC86   Wenn Übertrag, dann Ende (Fehler!)
CC84:  68          PLA          Hole Zeile von Stack
CC85:  60          RTS          Rücksprung aus dem Unterprogramm

******************************  Eingabezeile klarmachen

CC86:  85 EC       STA  * $EC   Sichern der aktuellen Cursor Spalte
CC88:  85 E9       STA  * $E9   Sichern der Starteingabezeile
CC8A:  68          PLA          Hole Zeile von Stack
CC8B:  85 EB       STA  * $EB   aktuelle Cursor Zeile zurückschreib.
CC8D:  85 E8       STA  * $E8   als Starteingabezeile merken
CC8F:  20 5C C1    JSR  $C15C   Adresse aktuelle Zeile ermitteln
CC92:  20 57 CD    JSR  $CD57   Setze Cursor auf aktuelle Spalte
CC95:  A5 EB       LDA  * $EB   Hole aktuelle Cursor Zeile in Akku
CC97:  E5 E5       SBC  * $E5   Subtrahiere obere Grenze Fenster
CC99:  AA          TAX          Ergebnis dann nach X
CC9A:  38          SEC          Setze Carry für Subtraktion
CC9B:  A5 EC       LDA  * $EC   Hole aktuelle Cursor Spalte in Akku
CC9D:  E5 E6       SBC  * $E6   Subtrahiere linke Fenstergrenze
CC9F:  A8          TAY          Ergebnis dann nach Y
CCA0:  18          CLC          Lösche Carry für OK
CCA1:  60          RTS          Rücksprung aus dem Unterprogramm

******************************  Einsprung Kernal: PFKEY
                                Programmieren einer Funktionstaste

CCA2:  CA          DEX          Nummer d. F-Taste um 1 vermindern
CCA3:  86 DC       STX  * $DC   Nummer der (F-Taste -1) in Z-Page
CCA5:  84 DA       STY  * $DA   Länge d. F-Strings in Z-Page sichern
CCA7:  8D AA 02    STA  $02AA   Z-Page Adr.d.Stringzeigers in FETVEC
```

```
CCAA:  A8        TAY              Z-Page Adr. des Stringzeigers in Y
CCAB:  B6 02     LDX   * $02,Y    Bank Nr. d. F-Strings in X-Reg holen
CCAD:  20 6B FF  JSR   $FF6B      Kernal GETCFG: Hole Config. Wert
CCB0:  85 DE     STA   * $DE      Sichere ihn in Bank Byte f. F-String
CCB2:  A2 0A     LDX   # $0A      Anzahl der F-Tasten (10) in Akku
CCB4:  20 20 CD  JSR   $CD20      Addiere F-Stringlängen bis (X -1)
CCB7:  85 DB     STA   * $DB      Gesamtstringlänge in Z-Page sichern
CCB9:  A6 DC     LDX   * $DC      Nr. der (F-Taste -1) zurückholen
CCBB:  E8        INX              Echte F-Tastennummer herstellen
CCBC:  20 20 CD  JSR   $CD20      Addiere F-Stringlängen bis (X -1)
CCBF:  85 DD     STA   * $DD      Stringlänge bis zur F-Taste sichern
CCC1:  A6 DC     LDX   * $DC      Nr. der (F-Taste -1) zurückholen
CCC3:  A5 DA     LDA   * $DA      Stringlänge der F-Taste zurückholen
CCC5:  38        SEC              Carry ein f. ordentliche Subtraktion
CCC6:  FD 00 10  SBC   $1000,X    Länge des alten F-Strings abziehen
CCC9:  F0 2B     BEQ   $CCF6      Keine Verschiebung notwendig, weiter
CCCB:  90 16     BCC   $CCE3      Neue Stringlänge kleiner alte Länge
CCCD:  18        CLC              Carry für Addition löschen
CCCE:  65 DB     ADC   * $DB      Gesamtlänge+Differenzlänge addieren
CCD0:  B0 4D     BCS   $CD1F      Länge größer 256, dann Fehler: RTS
CCD2:  AA        TAX              Neue Maximallänge in X-Reg ablegen
CCD3:  A4 DB     LDY   * $DB      Alte Maximallänge in Y-Reg holen
CCD5:  C4 DD     CPY   * $DD      Sind beide Längen gleich, dann ist
CCD7:  F0 1D     BEQ   $CCF6      die letzte F-Taste angesprochen
CCD9:  88        DEY              Alte Maximallänge um 1 vermindern
CCDA:  CA        DEX              Neue Maximallänge um 1 vermindern
CCDB:  B9 0A 10  LDA   $100A,Y    Alle F-Tasten Strings bis zur neuen
CCDE:  9D 0A 10  STA   $100A,X    Einschubposition nach hinten ziehen
CCE1:  B0 F2     BCS   $CCD5      und Platz für neuen String schaffen.
CCE3:  65 DD     ADC   * $DD      Differenzlänge zur Bislänge addieren
CCE5:  AA        TAX              Neue Bislänge in X-Reg kopieren
CCE6:  A4 DD     LDY   * $DD      Alte Bislänge in Y-Reg holen
CCE8:  C4 DB     CPY   * $DB      Mit alter Maximallänge vergleichen
CCEA:  B0 0A     BCS   $CCF6      Gleich, dann ist das Heranziehen
CCEC:  B9 0A 10  LDA   $100A,Y    der F-Tastenstrings ab der neuen
CCEF:  9D 0A 10  STA   $100A,X    F-Taste bis zum F-Tastenstringende
CCF2:  C8        INY              abgeschlossen. Alte + neue Bislänge
CCF3:  E8        INX              für Verschiebung um 1 erhöhen.
CCF4:  90 F2     BCC   $CCE8      Schleifen, bis F-Strings verschoben

*******************************   Neuen F-Tastenstring einsetzen

CCF6:  A6 DC     LDX   * $DC      Nr. der (F-Taste -1) zurückholen
CCF8:  20 20 CD  JSR   $CD20      Addiere F-Stringlängen bis (X -1)
CCFB:  AA        TAX              Stringlänge bis zur neuen F-Taste
CCFC:  A4 DC     LDY   * $DC      Nr. der (F-Taste -1) zurückholen
CCFE:  A5 DA     LDA   * $DA      Länge des einzusetzenden F-Strings
CD00:  99 00 10  STA   $1000,Y    Längeneintrag in F-Tabelle ersetzen
```

```
CD03:  A0 00     LDY  # $00     Displacementzeiger initialisieren
CD05:  C6 DA     DEC  * $DA     Länge des F-Strings = Länge -1
CD07:  30 15     BMI  $CD1E     Alle Zeichen in Tab. übertragen,Exit
CD09:  86 DF     STX  * $DF     Die "Bis" Stringlänge sichern
CD0B:  A6 DE     LDX  * $DE     Bankwert, wo neuer F-String steht
CD0D:  AD AA 02  LDA  $02AA     Akku mit FETVEC laden
CD10:  78        SEI            Alle System Interrupts verhindern
CD11:  20 A2 02  JSR  $02A2     FETCH: F-String Zeichen holen
CD14:  58        CLI            Alle System Interrupts freigeben
CD15:  A6 DF     LDX  * $DF     Position,f. F-String Eintrag in Tab.
CD17:  9D 0A 10  STA  $100A,X   Zeichen in F-String Tab. eintragen
CD1A:  E8        INX            Displ. auf "Wohin" String-Puffer +1
CD1B:  C8        INY            Displ. auf "Woher" String-Puffer +1
CD1C:  D0 E7     BNE  $CD05     Sprung i.d. String Transfer Schleife
CD1E:  18        CLC            Kennzeichen für "OK" Rücksprung
CD1F:  60        RTS            Rücksprung aus dem Unterprogramm

******************************  Addiere Länge aller F-Strings bis X

CD20:  A9 00     LDA  # $00     Lade Zähler mit Null
CD22:  18        CLC            Lösche Carry zur Addition
CD23:  CA        DEX            Vorangehende Key-Belegung
CD24:  30 05     BMI  $CD2B     Wenn Null, dann alle addiert
CD26:  7D 00 10  ADC  $1000,X   Addiere Länge von Key X
CD29:  90 F8     BCC  $CD23     Und springe unbedingt nach $CD23
CD2B:  60        RTS            Rücksprung aus dem Unterprogramm

******************************  Kernal Routine: SWAPPER
                                Umschalten von 40/80-Zeichen-Modus

CD2C:  85 F0     STA  * $F0     Speichere Akku als zuletzt gedruck-
CD2E:  A2 1A     LDX  # $1A     tes Zeichen und tausche dann den
CD30:  BC 40 0A  LDY  $0A40,X   passiven Monitor-Speicher  mit dem
CD33:  B5 E0     LDA  * $E0,X   passiven Speicher aus. Dies ge-
CD35:  9D 40 0A  STA  $0A40,X   schieht 26mal, da 26 Bytes zu
CD38:  98        TYA            kopieren sind. Der passive Bereich
CD39:  95 E0     STA  * $E0,X   liegt im Bereich $0A40 bis $0A5B.
CD3B:  CA        DEX            Erniedrige den Zähler und springe,
CD3C:  10 F2     BPL  $CD30     wenn noch nicht alles ausgetauscht
CD3E:  A2 0D     LDX  # $0D     Jetzt müssen noch die Bitmaps, die
CD40:  BC 60 0A  LDY  $0A60,X   Bittabellen, von aktivem und pas-
CD43:  BD 54 03  LDA  $0354,X   sivem Bildschirm ausgetauscht
CD46:  9D 60 0A  STA  $0A60,X   werden. Dies geschieht 13mal
CD49:  98        TYA            Der passive Bereich beginnt bei
CD4A:  9D 54 03  STA  $0354,X   $0A60.
CD4D:  CA        DEX            Erniedrige Zähler und springe,
CD4E:  10 F0     BPL  $CD40     wenn noch nicht alles kopiert.
CD50:  A5 D7     LDA  * $D7     Hole Status 40/80-Zeichen
```

```
CD52:  49 80     EOR  # $80     und kehre Flag-Bit um
CD54:  85 D7     STA  * $D7     wieder abspeichern
CD56:  60        RTS            Rücksprung aus dem Unterprogramm

*****************************   Setze Cursor auf aktuelle Spalte

CD57:  24 D7     BIT  * $D7     Teste auf 40/80-Zeichen-Modus
CD59:  10 FB     BPL  $CD56     Bei 40-Zeichen-Modus Ende
CD5B:  A2 0E     LDX  # $0E     Cursor-Position High
CD5D:  18        CLC            Lösche Carry
CD5E:  A5 E0     LDA  * $E0     Lo-Byte von akt. Bildschirmzeile
CD60:  65 EC     ADC  * $EC     Addiere Cursor-Spalte
CD62:  48        PHA            Rette Lo-Byte
CD63:  A5 E1     LDA  * $E1     Hi-Byte von akt. Bildschirmzeile
CD65:  69 00     ADC  # $00     Addiere den Übertrag
CD67:  20 CC CD  JSR  $CDCC     Und speichere Hi-Byte ab
CD6A:  E8        INX            Erhöhe Registerzeiger auf $0F
CD6B:  68        PLA            Hole Lo-Byte von Stack
CD6C:  4C CC CD  JMP  $CDCC     Und auch abspeichern (Rücksprung)

*****************************   Cursorfarbe an Cursorpos. setzen

CD6F:  24 D7     BIT  * $D7     Teste auf 40/80-Zeichen-Modus
CD71:  10 26     BPL  $CD99     Springe wenn 40-Zeichen-Modus
CD73:  20 7C C1  JSR  $C17C     Attributadresse setzen
CD76:  A4 EC     LDY  * $EC     Hole aktuelle Cursor Spalte in Y-Reg
CD78:  20 F9 CD  JSR  $CDF9     Attribut-Adresse in Update-Register
CD7B:  20 D8 CD  JSR  $CDD8     Hole aktuelles Attribut
CD7E:  8D 33 0A  STA  $0A33     Zwischenspeichern
CD81:  29 F0     AND  # $F0     Bits 0-3 ausmaskieren (Farbe)
CD83:  85 DB     STA  * $DB     und zwischenspeichern
CD85:  20 F9 CD  JSR  $CDF9     Attribut-Adresse in Update-Register
CD88:  A5 F1     LDA  * $F1     Farbcode für Zeichenausgabe in Akku
CD8A:  29 0F     AND  # $0F     Bits 4-7 ausmaskieren (Attribut)
CD8C:  05 DB     ORA  * $DB     und mit Attribut verknüpfen
CD8E:  20 CA CD  JSR  $CDCA     Abspeichern an Attribut-Adresse
CD91:  A2 0A     LDX  # $0A     Cursor-Modus und Start-Scan-Line
CD93:  AD 2B 0A  LDA  $0A2B     80-Zeichen-Cursor-Modus
CD96:  4C CC CD  JMP  $CDCC     Und abspeichern
CD99:  A9 00     LDA  # $00     Akku gleich Null und abspeichern
CD9B:  8D 27 0A  STA  $0A27     bedeutet VIC-Cursor ausschalten
CD9E:  60        RTS            Rücksprung aus dem Unterprogramm

*****************************   Cursor einschalten (80er)

CD9F:  24 D7     BIT  * $D7     Teste 40/80-Zeichen-Modus
CDA1:  10 10     BPL  $CDB3     Springe, wenn 40-Zeichen-Modus
CDA3:  20 F9 CD  JSR  $CDF9     Setze Update auf Attribut-Adresse
```

```
CDA6:  AD 33 0A   LDA  $0A33     Zwischenspeicher für MOVLIN
CDA9:  20 CA CD   JSR  $CDCA     Attribut speichern
CDAC:  A2 0A      LDX  # $0A     Cursor-Modus und Start-Scan-Line
CDAE:  A9 20      LDA  # $20     werden mit dem Wert 32 belegt.
CDB0:  4C CC CD   JMP  $CDCC     Lege Akku in VDC-Data-Register ab

******************************   Cursor einschalten (40er)

CDB3:  8D 27 0A   STA  $0A27     VIC-Cursor einschalten
CDB6:  AD 26 0A   LDA  $0A26     Starrer oder blinkender Cursor?
CDB9:  10 0E      BPL  $CDC9     Starrer, dann Ende
CDBB:  29 40      AND  # $40     Lösche Blink-Flag
CDBD:  8D 26 0A   STA  $0A26     und wieder abspeichern
CDC0:  AD 29 0A   LDA  $0A29     VIC-Zeichen vor Blinken
CDC3:  AE 2A 0A   LDX  $0A2A     VIC-Farbe vor Blinken
CDC6:  20 34 CC   JSR  $CC34     Setze alte Werte
CDC9:  60         RTS            Rücksprung aus dem Unterprogramm

******************************   Akku ins Data-Register von VCR

CDCA:  A2 1F      LDX  # $1F     Data-Register von VCR
CDCC:  8E 00 D6   STX  $D600     Register übermitteln
CDCF:  2C 00 D6   BIT  $D600     Teste Status -
CDD2:  10 FB      BPL  $CDCF     Noch nicht fertig, warten!
CDD4:  8D 01 D6   STA  $D601     Speichere Wert ins Register
CDD7:  60         RTS            Rücksprung aus dem Unterprogramm

******************************   Hole Wert des Data-Registers

CDD8:  A2 1F      LDX  # $1F     Data-Register von VCR
CDDA:  8E 00 D6   STX  $D600     Register übermitteln
CDDD:  2C 00 D6   BIT  $D600     Teste Status
CDE0:  10 FB      BPL  $CDDD     Noch nicht fertig, warten!
CDE2:  AD 01 D6   LDA  $D601     Hole Wert des Registers
CDE5:  60         RTS            Rücksprung aus dem Unterprogramm

******************************   Setze Update-Adresse auf aktuelle
                                 Bildschirmposition

CDE6:  A2 12      LDX  # $12     Update-Adresse Hi
CDE8:  18         CLC            Lösche Carry für Addition
CDE9:  98         TYA            Y (Spalte) in Akku
CDEA:  65 E0      ADC  * $E0     und Lo-Byte von akt. Adresse
CDEC:  48         PHA            addieren, dann auf Stack
CDED:  A9 00      LDA  # $00     Akku mit Null laden um dann
CDEF:  65 E1      ADC  * $E1     den Übertrag zu addieren
CDF1:  20 CC CD   JSR  $CDCC     Speichern des Hi-Bytes
CDF4:  68         PLA            Hole Lo-Byte von Stack
```

```
CDF5:  E8        INX           Erhöhe Register auf $13
CDF6:  4C CC CD  JMP  $CDCC    und Lo-Byte in Update-Register

******************************  Setze Update-Adresse für Attribut

CDF9:  A2 12     LDX  # $12    Update-Register Hi-Byte
CDFB:  18        CLC           Lösche Carry für Addition
CDFC:  98        TYA           Y (Spalte) nach Akku
CDFD:  65 E2     ADC  * $E2    Addiere Lo-Byte von Attribut-
CDFF:  48        PHA           Adresse und dann auf Stack
CE00:  A9 00     LDA  # $00    Lade Akku mit Null, um den
CE02:  65 E3     ADC  * $E3    Übertrag zu addieren
CE04:  20 CC CD  JSR  $CDCC    Abspeichern des Hi-Bytes
CE07:  68        PLA           Hole Lo-Byte von Stack und
CE08:  E8        INX           Erhöhe Register auf $13
CE09:  4C CC CD  JMP  $CDCC    Speicher Lo-Byte ab

******************************  Zeichensatz in VDC-RAM kopieren

CE0C:  A9 00     LDA  # $00    Lade Akku (Low) und Y (High) mit
CE0E:  A0 D0     LDY  # $D0    Startadresse vom CHARROM: $D000
CE10:  85 DA     STA  * $DA    Speicher diese Werte in Zeropage-
CE12:  84 DB     STY  * $DB    Adresse $DA und $DB
CE14:  A2 12     LDX  # $12    Update-Register Hi
CE16:  A9 20     LDA  # $20    Startadresse des Zeichengenerators
CE18:  20 CC CD  JSR  $CDCC    im VDC definieren
CE1B:  E8        INX           Zeiger nun auf Lo-Byte
CE1C:  A9 00     LDA  # $00    $00 ist Lo-Byte der Startadresse
CE1E:  20 CC CD  JSR  $CDCC    des Zeichengenerators
CE21:  A0 00     LDY  # $00    Index-Zeiger auf Zeile/Zeichen
CE23:  A2 0E     LDX  # $0E    CHARROM auswählen
CE25:  A9 DA     LDA  # $DA    anzusprechende Zeropage-Adresse
CE27:  20 74 FF  JSR  $FF74    Kernal INDFET: LDA(XX),Y v.bel. Bank
CE2A:  20 CA CD  JSR  $CDCA    und abspeichern des Wertes im RAM
CE2D:  C8        INY           VDC. Index-Zeiger erhöhen
CE2E:  C0 08     CPY  # $08    Schon alle 8 Zeilen kopiert?
CE30:  90 F1     BCC  $CE23    Nein, dann nächste Zeile
CE32:  A9 00     LDA  # $00    Sonst lade Akku mit Null
CE34:  20 CA CD  JSR  $CDCA    und speichere diesen Wert im VDC-RAM
CE37:  88        DEY           achtmal ab.
CE38:  D0 FA     BNE  $CE34    Springe, wenn noch nicht fertig
CE3A:  18        CLC           Lösche Carry für Addition
CE3B:  A5 DA     LDA  * $DA    Lade Akku mit Lo-Byte
CE3D:  69 08     ADC  # $08    und addiere 8 hinzu
CE3F:  85 DA     STA  * $DA    Wieder abspeichern und
CE41:  90 E0     BCC  $CE23    wenn kein Übertrag, dann weiter
CE43:  E6 DB     INC  * $DB    sonst Übertrag berücksichtigen
CE45:  A5 DB     LDA  * $DB    und überprüfen, ob das Hi-Byte
```

```
CE47:  C9 E0       CMP  # $E0      bereits aufs Ende des CHARROM
CE49:  90 D8       BCC  $CE23      zeigt, sonst weitermachen
CE4B:  60          RTS             Rücksprung aus dem Unterprogramm

*****************************      Tabelle der Farbcodes (ASCII)

CE4C:  90 05 1C 9F 9C 1E 1F 9E
CE54:  81 95 96 97 98 99 9A 9B

*****************************      Tabelle der Farbcodes für VDC

CE5C:  00 0F 08 07 0B 04 02 0D
CE64:  0A 0C 09 06 01 05 03 0E

*****************************      Zweierpotenzen

CE6C:  80 40 20 10 08 04 02 01     128, 64, 32, 16, 8, 4, 2, 1

*****************************      Initialisierungswerte 40 Z. Schirm

CE74:  00 04 00 D8 18 00 00 27     Diese Werte werden bei der
CE7C:  00 00 00 00 00 18 27 00     Initialisierung in die Zero-Page
CE84:  00 0D 0D 00 00 00 00 00     ab $E0 kopiert. Die Bedeutung
CE8C:  00 00                       entnehmen Sie der Z-Page Belegung

*****************************      Initialisierungswerte 80 Z. Schirm

CE8E:  00 00 00 08 18 00 00 4F     Diese Werte werden bei der
CE96:  00 00 00 00 00 18 4F 00     Initialisierung in die Page 3 ab
CE9E:  00 07 07 00 00 00 00 00     Adr. $0A40 kopiert. Die Bedeutung
CEA6:  00 00                       entnehmen Sie der Page-3 Belegung

*****************************      Einschaltlängenbelegung der F-Tasten

CEA8:07060A0706040508  Längeder Funktionstastenstrings
CEB0:  09 05                       (F1 - F8, Shift-Run, Help)

*****************************      Einschaltstringbelegung der F-Tasten

CEB2:  47 52 41 50 48 49 43        GRAPHIC
CEB9:  44 4C 4F 41 44 22           DLOAD"
CEBF:  44 49 52 45 43 54 4F 52     DIRECTORY  <Cr>
CEC7:  59 0D
CEC9:  53 43 4E 43 4C 52 0D        SCNCLR  <Cr>
CED0:  44 53 41 56 45 22           DSAVE"
CED6:  52 55 4E 0D                 RUN  <Cr>
CEDA:  4C 49 53 54 0D              LIST  <Cr>
CEDF:  4D 4F 4E 49 54 4F 52 0D     MONITOR  <Cr>
```

```
CEE7:  44 CC 22 2A 0D 52 55 4E    D <Shift - L>  <Cr>  RUN  <Cr>
CEEF:  0D
CEF0:  48 45 4C 50 0D             HELP  <Cr>

******************************    Freier Bereich

CEF5:  FF FF FF . . .             Nicht benutzt
CFFD:  . . . FF 00 FF             Nicht benutzt

**************************************************************************

******************************    Reset Routine

E000:  A2 FF     LDX  # $FF       Initialisierungswert f. Stackpointer
E002:  78        SEI              Alle System Interrupts verhindern
E003:  9A        TXS              System Stackpointer a. Anfang setzen
E004:  D8        CLD              Dezimale Betriebsart zurücksetzen
E005:  A9 00     LDA  # $00       Akku mit $00 laden und alle
E007:  8D 00 FF  STA  $FF00       System ROMs einschalten
E00A:  A2 0A     LDX  # $0A       Schleifen und Displ. Zähler setzen
E00C:  BD 4B E0  LDA  $E04B,X     Byte aus Initialisierungs-Tab. holen
E00F:  9D 00 D5  STA  $D500,X     MMU Register initialisieren
E012:  CA        DEX              Schleifen und Displ. Zähler - 1
E013:  10 F7     BPL  $E00C       11 Werte aus Tabelle übertragen
E015:  8D 04 0A  STA  $0A04       NMI/Reset Status Zeiger löschen
E018:  20 CD E0  JSR  $E0CD       NMI, IRQ + Z-Page Routinen kopieren
E01B:  20 F0 E1  JSR  $E1F0       Prüfe auf <CBM> Code in RAM 1
E01E:  20 42 E2  JSR  $E242       Modultest für C-64 Konfiguration
E021:  20 09 E1  JSR  $E109       Kernal IOINIT: Init. I/O Einheiten
E024:  20 3D F6  JSR  $F63D       Shift RUN/STOP Tastaturabfrage
Q027:  48        PHA              Akku Inhalt auf Stack retten
E028:  30 07     BMI  $E031       Bit 7 gesetzt,Skip Reset Status Test
E02A:  A9 A5     LDA  # $A5       System Warm-/Kaltstart Statuszeiger
E02C:  CD 02 0A  CMP  $0A02       auf Warmstart Status testen
E02F:  F0 03     BEQ  $E034       Warmstart Status, dann Skip
E031:  20 93 E0  JSR  $E093       Kernal RAMTAS: RAM löschen / testen
E034:  20 56 E0  JSR  $E056       Kernal RESTOR: I/O initialisieren
E037:  20 00 C0  JSR  $C000       Routine CINT: Init. Editor + Screen
E03A:  68        PLA              Code der Tastaturabfrage zurückholen
E03B:  58        CLI              Alle System Interrupts freigeben
E03C:  30 03     BMI  $E041       Bit 7 gesetzt,Skip Monitor Einsprung
E03E:  4C 00 B0  JMP  $B000       Kernal MONITOR Einsprung
E041:  C9 DF     CMP  # $DF       System als C-64 konfigurieren?
E043:  F0 03     BEQ  $E048       Ja,dann konfiguriere System als C-64
E045:  6C 00 0A  JMP  ($0A00)     System Restart Vektor (normal $4003)
E048:  4C 4B E2  JMP  $E24B       Kernal GO64MODE: Konfiguriere C-64
```

```
******************************     Initialisierungstabelle für MMU

E04B:  00        .Byte  $00        $D500: Configuration Register
E04C:  00        .Byte  $00        $D501: Preconfiguration Register A
E04D:  00        .Byte  $00        $D502: Preconfiguration Register B
E04E:  00        .Byte  $00        $D503: Preconfiguration Register C
E04F:  00        .Byte  $00        $D504: Preconfiguration Register D
E050:  BF        .Byte  $BF        $D505: Mode Configuration Register
E051:  04        .Byte  $04        $D506: RAM Configuration Register
E052:  00        .Byte  $00        $D507: Page 0 Zeiger Low
E053:  00        .Byte  $00        $D508: Page 0 Zeiger High
E054:  01        .Byte  $01        $D509: Page 1 Zeiger Low
E055:  00        .Byte  $00        $D50A: Page 1 Zeiger High

******************************     Kernal Routine: RESTOR

E056:  A2 73     LDX  # $73        Lo-Byte der Kernal Vektortabelle
E058:  A0 E0     LDY  # $E0        Hi-Byte der Kernal Vektortabelle
E05A:  18        CLC               KZ für Download der Vektortabelle

******************************     Kernal Routine: VEKTOR

E05B:  86 C3     STX  * $C3        Lo-Byte der Vektortabelle in Z-Page
E05D:  84 C4     STY  * $C4        Hi-Byte der Vektortabelle  ($E073)
E05F:  A0 1F     LDY  # $1F        Schleifenzähler auf 32 setzen
E061:  B9 14 03  LDA  $0314,Y      Byte aus Page 3 Vektortabelle lesen
E064:  B0 02     BCS  $E068        Wenn Upload der Vektortabelle, Skip
E066:  B1 C3     LDA  ($C3),Y      Einen Wert aus Vektortabelle lesen
E068:  99 14 03  STA  $0314,Y      In Page 3 Vektortabelle speichern
E06B:  90 02     BCC  $E06F        Wenn Download der Vektortabelle,Skip
E06D:  91 C3     STA  ($C3),Y      In indizierte Tabelle kopieren
E06F:  88        DEY               Schleifenzähler und Displacement -1
E070:  10 EF     BPL  $E061        Schleifen bis Tabelle übertragen
E072:  60        RTS               Rücksprung aus dem Unterprogramm

******************************     Vektortabelle

E073:  65 FA     $FA65             Vektor zeigt auf IRQ-Einsprung
E075:  03 B0     $B003             Vektor auf Monitor-Break-Einsprung
E077:  40 FA     $FA40             Vektor zeigt auf NMI-Einsprung
E079:  BD EF     $EFBD             Vektor zeigt auf Kernal OPEN  Rout.
E07B:  88 F1     $F188             Vektor zeigt auf Kernal CLOSE Rout.
E07D:  06 F1     $F106             Vektor zeigt auf Kernal CHKIN Rout.
E07F:  4C F1     $F14C             Vektor zeigt auf Kernal CKOUT Rout.
E081:  26 F2     $F226             Vektor zeigt auf Kernal CLRCH Rout.
E083:  06 EF     $EF06             Vektor zeigt auf Kernal BASIN Rout.
E085:  79 EF     $EF79             Vektor zeigt auf Kernal BSOUT Rout.
E087:  6E F6     $F66E             Vektor zeigt auf Kernal STOP  Rout.
```

```
E089:  EB EE     $EEEB          Vektor zeigt auf Kernal GETIN Rout.
F08B:  22 F2     $F222          Vektor zeigt auf Kernal CLALL Rout.
E08D:  06 B0     $B006          Vektor auf Monitor Exmon Einsprung
E08F:  6C F2     $F26C          Vektor zeigt auf LOAD Einsprung
E091:  4E F5     $F54E          Vektor zeigt auf SAVE Einsprung

*****************************   Kernal Routine: RAMTAS
                                Lösche Z-Page, setze Memtop, Membot,
                                RS-232 E-/A-Buffer + Kassettenpuffer

E093:  A9 00     LDA  # $00     Akku mit $00 für Adreßwerte Lo
E095:  A8        TAY            initialisieren und nach Y kopieren
E096:  99 02 00  STA  $0002,Y   Löschen der gesamten Zero-Page
E099:  C8        INY            mit Ausnahme der beiden Prozessor-
E09A:  D0 FA     BNE  $E096     portregister $00 und $01
E09C:  A0 0B     LDY  # $0B     Setze den Zero-Page Kassettenpuffer-
E09E:  84 B3     STY  * $B3     zeiger (Z-Page $B2-$B3) auf die
E0A0:  85 B2     STA  * $B2     Anfangsadresse $0B00
E0A2:  A0 0C     LDY  # $0C     Setze den Zero-Page RS-232 Eingabe-
E0A4:  84 C9     STY  * $C9     pufferzeiger (Z-Page $C8-$C9) auf
E0A6:  85 C8     STA  * $C8     die Anfangsadresse $0C00
E0A8:  A0 0D     LDY  # $0D     Setze den Zero-Page RS-232 Ausgabe-
E0AA:  84 CB     STY  * $CB     pufferzeiger (Z-Page $CA-$CB) auf
E0AC:  85 CA     STA  * $CA     die Anfangsadresse $0D00
E0AE:  18        CLC            Carry Flag als Kennzeichen löschen
E0AF:  A0 FF     LDY  # $FF     Setze die Speicherobergrenze
E0B1:  A2 00     LDX  # $00     in der Systembank auf $FF00
E0B3:  20 6B F7  JSR  $F76B     Sprung in Kernal Routine MEMTOP
E0B6:  A0 1C     LDY  # $1C     Setze die Speicheruntergrenze
E0B8:  A2 00     LDX  # $00     in der Systembank auf $1C00
E0BA:  20 7A F7  JSR  $F77A     Sprung in Kernal Routine MEMBOT
E0BD:  A0 40     LDY  # $40     Initialisiere den System
E0BF:  A2 00     LDX  # $00     RESTART Vektor an der Adresse
E0C1:  8C 01 0A  STY  $0A01     $A00-$A01 mit dem Wert $4000 für
E0C4:  8E 00 0A  STX  $0A00     den System Kaltstart Einsprung
E0C7:  A9 A5     LDA  # $A5     Initialisiere den System Kaltstart/
E0C9:  8D 02 0A  STA  $0A02     Warmstart Statuszeiger mit $A5
E0CC:  60        RTS            Rücksprung aus dem Unterprogramm

*****************************   NMI, IRQ + Z-Page Routinen kopieren

E0CD:  A0 03     LDY  # $03     Schleifenzähler f. 4 Schleifen init.
E0CF:  B9 05 E1  LDA  $E105,Y   Wert aus RAM-Bank Tabelle holen
E0D2:  8D 00 FF  STA  $FF00     Entsprechende Konfiguration setzen
E0D5:  A2 3F     LDX  # $3F     64 Bytes sind zu übertragen
E0D7:  BD 05 FF  LDA  $FF05,X   NMI + IRQ Routine aus ROM auslesen
E0DA:  9D 05 FF  STA  $FF05,X   In darunterliegendes RAM kopieren
E0DD:  CA        DEX            Übertragungsschleifenzähler -1
```

```
E0DE:  10 F7     BPL  $E0D7       Schleifen, bis 64 Bytes kopiert sind
E0E0:  A2 05     LDX  # $05       In gleicher Weise werden hier der
E0E2:  BD FA FF  LDA  $FFFA,X     NMI, Reset und IRQ Vektor aus dem
E0E5:  9D FA FF  STA  $FFFA,X     Kernal ROM in das darunterliegende
E0E8:  CA        DEX              RAM kopiert. Schleifen, bis alle 3
E0E9:  10 F7     BPL  $E0E2       Vektoren übertragen sind.
E0EB:  88        DEY              Schleifenzähler f. 4 RAM-Banks -1
E0EC:  10 E1     BPL  $E0CF       Rout.+Vekt. in 4 RAM-Banks kopieren
E0EE:  A2 59     LDX  # $59       90 Bytes sind zu übertragen
E0F0:  BD 00 F8  LDA  $F800,X     Hier werden die ROM Originale der
E0F3:  9D A2 02  STA  $02A2,X     FETCH, STASH, CMPARE, JSRFAR und
E0F6:  CA        DEX              JMPFAR Routinen in den RAM Bereich
E0F7:  10 F7     BPL  $E0F0       der Page 2 und 3 kopiert.
E0F9:  A2 0C     LDX  # $0C       13 Bytes sind zu übertragen
E0FB:  BD 5A F8  LDA  $F85A,X     Hier wird die im ROM als
E0FE:  9D F0 03  STA  $03F0,X     Original abgelegte Routine
E101:  CA        DEX              in den RAM Bereich ab Adresse
E102:  10 F7     BPL  $E0FB       $03F0 kopiert.
E104:  60        RTS              Rücksprung aus dem Unterprogramm

*****************************     RAM-Bank Tabelle

E105:  00        .Byte  $00       RAM 0,Syst.ROM,Basic Hi,Basic Lo,I/O
E106:  40        .Byte  $40       RAM 1,Syst.ROM,Basic Hi,Basic Lo,I/O
E107:  80        .Byte  $80       RAM 2,Syst.ROM,Basic Hi,Basic Lo,I/O
E108:  C0        .Byte  $C0       RAM 3,Syst.ROM,Basic Hi,Basic Lo,I/O

*****************************     Kernal Routine: IOINIT
                                  Initialisierung der CIAs

E109:  A9 7F     LDA  # $7F       Wert für "Interrupt löschen" laden
E10B:  8D 0D DC  STA  $DC0D       ICR des CIA 1 initialisieren
E10E:  8D 0D DD  STA  $DD0D       ICR des CIA 2 initialisieren
E111:  8D 00 DC  STA  $DC00       Port A, CIA 1, Matrixzeile 0
E114:  A9 08     LDA  # $08       "one shot" Initialisierung für Timer
E116:  8D 0E DC  STA  $DC0E       CRA des CIA 1 Timer A auf "one shot"
E119:  8D 0E DD  STA  $DD0E       CRA des CIA 2 Timer A auf "one shot"
E11C:  8D 0F DC  STA  $DC0F       CRA des CIA 1 Timer B auf "one shot"
E11F:  8D 0F DD  STA  $DD0F       CRA des CIA 2 Timer B auf "one shot"
E122:  A2 00     LDX  # $00       CIA Register auf Eingangs-Modus
E124:  8E 03 DC  STX  $DC03       Datenrichtungsregister B des CIA 1
E127:  8E 03 DD  STX  $DD03       Datenrichtungsregister B des CIA 2
E12A:  CA        DEX              X-Reg auf Wert für "Ausgabe-Modus"
E12B:  8E 02 DC  STX  $DC02       Datenrichtungsregister A des CIA 1
E12E:  A9 07     LDA  # $07       Videokontroller auf untere 16 K
E130:  8D 00 DD  STA  $DD00       ATN Signal am Port A , CIA 2 löschen
E133:  A9 3F     LDA  # $3F       Bit 0 bis 5 auf Ausgabe setzen
E135:  8D 02 DD  STA  $DD02       Datenrichtungsregister A des CIA 2
```

```
E138:  A9 E3     LDA  # $E3     Prozessorport Datenregister mit
E13A:  85 01     STA  * $01     Standardwert $E3 initialisieren
E13C:  A9 2F     LDA  # $2F     Prozessorport Datenrichtungsregister
E13E:  85 00     STA  * $00     mit Standardwert $2F initialisieren
E140:  A2 FF     LDX  # $FF     PAL/NTSC Zeiger initialisieren (PAL)
E142:  AD 11 D0  LDA  $D011     Warten bis das MSB des Rasterzeilen-
E145:  10 FB     BPL  $E142     Interrupt-Zeigers gesetzt ist
E147:  A9 08     LDA  # $08     Vergleichswert für PAL/NTSC-Version
E149:  CD 12 D0  CMP  $D012     Mit Lo Byte Raster-Interrupt vgl.
E14C:  90 06     BCC  $E154     Kleiner als 8, dann PAL Version
E14E:  AD 11 D0  LDA  $D011     Warten, bis das MSB des Rasterzeilen-
E151:  30 F4     BMI  $E147     Interrupts gelöscht ist.
E153:  E8        INX            PAL/NTSC-Zeiger auf NTSC setzen ($0)
E154:  8E 03 0A  STX  $0A03     PAL/NTSC-Versionszeiger sichern
E157:  A9 00     LDA  # $00     Initialisierungswert f. Zeiger
E159:  8D 37 0A  STA  $0A37     X-Reg Speicher für Bank Operationen
E15C:  8D 39 0A  STA  $0A39     80 Zeichen VDC Hilfsspeicher
E15F:  8D 0A 0A  STA  $0A0A     Indirekter IRQ Vektor (Kassette)
E162:  8D 3A 0A  STA  $0A3A     IRQ Hilfszeiger initialisieren
E165:  8D 36 0A  STA  $0A36     Rasterzeile für R-Interrupt auslösen
E168:  85 99     STA  * $99     Standard Eingabegerät = Tastatur
E16A:  A9 03     LDA  # $03     Z-Page Speicher f. Standard Ausgabe-
E16C:  85 9A     STA  * $9A     gerät auf 3 (= Bildschirm) setzen
E16E:  A2 30     LDX  # $30     49 Bytes sind zu übertragen
E170:  BD C7 E2  LDA  $E2C7,X   Initialisierungstabelle für VIC Chip
E173:  9D 00 D0  STA  $D000,X   in Vic Steuerregister kopieren
E176:  CA        DEX            Schleifen-/Displacement- Zähler -1
E177:  10 F7     BPL  $E170     Schleifen, bis 49 Werte übertragen
E179:  A2 00     LDX  # $00     Schleifenzähler für VDC Init. setzen
E17B:  20 DC E1  JSR  $E1DC     Initialisiere VDC Register
E17E:  AD 00 D6  LDA  $D600     VDC Status abfragen
E181:  29 07     AND  # $07     Prüfe, ob Bits 0-2 gelöscht sind
E183:  F0 05     BEQ  $E18A     Ja, Skip Initialisierung d. VDC-Reg.
E185:  A2 3B     LDX  # $3B     Displacementzeiger auf VDC Tab.
E187:  20 DC E1  JSR  $E1DC     Initialisiere VDC Register
E18A:  2C 03 0A  BIT  $0A03     Prüfe auf PAL/NTSC Version
E18D:  10 05     BPL  $E194     Skip, wenn NTSC Version
E18F:  A2 3E     LDX  # $3E     Displacementzeiger auf VDC Tab.
E191:  20 DC E1  JSR  $E1DC     Initialisiere VDC Register
E194:  AD 04 0A  LDA  $0A04     Prüfe NMI/Reset Status Zeiger
E197:  30 15     BMI  $E1AE     VDC schon initialisiert, dann Skip
E199:  20 27 C0  JSR  $C027     Routine INIT80: Initi. 80 Zeichen
E19C:  A9 80     LDA  # $80     Bit 7 im Akku setzen, diesen Wert
E19E:  0D 04 0A  ORA  $0A04     mit dem NMI/VDC Status verknüpfen
E1A1:  8D 04 0A  STA  $0A04     und in Status Flag zurückschreiben
E1A4:  A2 FF     LDX  # $FF     Schleifenzähler Hi auf High-Value
E1A6:  A0 FF     LDY  # $FF     Schleifenzähler Lo auf High-Value
E1A8:  88        DEY            Schleifenzähler Lo um 1 vermindern
```

```
E1A9:  D0 FD     BNE  $E1A8       Schleife Lo fertig? Nein, weiter
E1AB:  CA        DEX              Schleifenzähler Hi um 1 vermindern
E1AC:  D0 FA     BNE  $E1A8       Schleife Hi fertig? Nein, weiter
E1AE:  A9 00     LDA  # $00       Initialisierungswert f. SID-Register
E1B0:  A2 18     LDX  # $18       SID Disp.- und Schleifenzähler
E1B2:  9D 00 D4  STA  $D400,X     SID Register löschen (Low-Value)
E1B5:  CA        DEX              Schleifen und Disp. Zähler -1
E1B6:  10 FA     BPL  $E1B2       Schleifen, bis 19 Reg. gelöscht sind
E1B8:  A2 01     LDX  # $01       X-Reg mit #1 vorladen
E1BA:  8E 1A D0  STX  $D01A       IRQ Maskenregister setzen
E1BD:  CA        DEX              X-Reg auf #0 herunterzählen
E1BE:  8E 1C 0A  STX  $0A1C       Fast seriell Mode Zeiger löschen
E1C1:  8E 0F 0A  STX  $0A0F       RS-232 NMI Status Register löschen
E1C4:  CA        DEX              X-Reg auf High-Value ($FF) setzen
E1C5:  8E 06 DC  STX  $DC06       Wert im Timer B Lo ablegen
E1C8:  8E 07 DC  STX  $DC07       Wert im Timer B Hi ablegen
E1CB:  A2 11     LDX  # $11       Code für "Force Load" und Timer A
E1CD:  8E 0F DC  STX  $DC0F       starten in das CIA Steueregister
E1D0:  20 C3 E5  JSR  $E5C3       Testroutine, ob der Fast Seriell
E1D3:  20 D6 E5  JSR  $E5D6       Modus am Diskettenlaufwerk erkannt
E1D6:  20 C3 E5  JSR  $E5C3       und beantwortet wird
E1D9:  4C 4E E5  JMP  $E54E       Clock Lo Signal und RTS

******************************    Initialisieren der VDC Register

E1DC:  BC F8 E2  LDY  $E2F8,X     Registerauswahl aus Tab. holen
E1DF:  30 0D     BMI  $E1EE       Endekriterium (Bit 7 = on) prüfen
E1E1:  E8        INX              Displacement auf VDC Tabelle +1
E1E2:  BD F8 E2  LDA  $E2F8,X     Register Schreibwert aus Tab. holen
E1E5:  E8        INX              Displacement auf VDC Tabelle +1
E1E6:  8C 00 D6  STY  $D600       VDC Register Auswahl-Port setzen
E1E9:  8D 01 D6  STA  $D601       VDC Register Daten-Port beschreiben
E1EC:  10 EE     BPL  $E1DC       Sprung zum Schleifenanfang
E1EE:  E8        INX              Displacement auf VDC Tabelle +1
E1EF:  60        RTS              Rücksprung aus dem Unterprogramm

******************************    Prüfe auf <CBM> Code in RAM 1

E1F0:  A2 F5     LDX  # $F5       Den 2 Byte Zero-Page Zeiger in den
E1F2:  A0 FF     LDY  # $FF       Adressen $C3 (Lo) - $C4 (Hi) mit
E1F4:  86 C3     STX  * $C3       der Anfangsadresse auf die Kernal-
E1F6:  84 C4     STY  * $C4       Vektortabelle initialisieren ($FFF5)
E1F8:  A9 C3     LDA  # $C3       FETVEC für die FETCH Routine auf den
E1FA:  8D AA 02  STA  $02AA       Anfang der Vektortabelle setzen
E1FD:  A0 02     LDY  # $02       Displacement für FETCH Routine
E1FF:  A2 7F     LDX  # $7F       Konfigurationscode (RAM 1 only)
E201:  20 A2 02  JSR  $02A2       FETCH Rout.: LDA von beliebiger Bank
E204:  D9 C4 E2  CMP  $E2C4,Y     Prüft auf Code <C> <B> <M>
```

```
E207:  D0 1B     BNE  $E224       Nicht gleich, dann Exit
E209:  88        DEY              Schleifen, bis die drei Kenn-
E20A:  10 F3     BPL  $E1FF       buchstaben überprüft sind
E20C:  A2 F8     LDX  # $F8       Den 2 Byte Zero-Page Zeiger in den
E20E:  A0 FF     LDY  # $FF       Adressen $C3 (Lo) - $C4 (Hi) mit
E210:  86 C3     STX  * $C3       mit der Adresse des Kernal C-128
E212:  84 C4     STY  * $C4       Mode Vektors initialisieren ($FFF8)
E214:  A0 01     LDY  # $01       Displacement für FETCH Routine
E216:  A2 7F     LDX  # $7F       Konfigurationscode (RAM 1 only)
E218:  20 A2 02  JSR  $02A2       FETCH Rout.: LDA von beliebiger Bank
E21B:  99 02 00  STA  $0002,Y     Einsprungadresse Hi - Lo in
E21E:  88        DEY              Z-Page $02-$03 bringen. Schleifen,
E21F:  10 F5     BPL  $E216       bis beide Adressen übertragen sind.
E221:  6C 02 00  JMP  ($0002)     Indirekter Sprung über Z-Page

******************************    Kernal Vektor: C128MODE

E224:  A9 40     LDA  # $40       RAM 1, alle System ROMs einschalten
E226:  8D 00 FF  STA  $FF00       und Konfiguration setzen
E229:  A9 24     LDA  # $24       Den 2 Byte Kernal Vektor
E22B:  A0 E2     LDY  # $E2       für den C-128 Mode mit
E22D:  8D F8 FF  STA  $FFF8       dem Standardwert
E230:  8C F9 FF  STY  $FFF9       $E224 initialisieren
E233:  A2 03     LDX  # $03       Schleifenzähler für 3 Übertragungen
E235:  BD C3 E2  LDA  $E2C3,X     <C> <B> <M> aus Tabelle laden
E238:  9D F4 FF  STA  $FFF4,X     und in den Vektorbereich der RAM 1
E23B:  CA        DEX              Bank kopieren. Schleifen, bis die
E23C:  D0 F7     BNE  $E235       drei Kennbuchstaben übertragen sind
E23E:  8E 00 FF  STX  $FF00       RAM 0, alle System ROMs einschalten
E241:  60        RTS              Rücksprung aus dem Unterprogramm

******************************    Prüfe,ob EXROM Eingang am MCR belegt
                                  Dient der Umschaltung bei
                                  eingesteckten 64er Modulen

E242:  AD 05 D5  LDA  $D505       Das MCR Register der MMU auslesen
E245:  29 30     AND  # $30       Prüfe, ob Bit 5 für den EXROM
E247:  C9 30     CMP  # $30       Eingang gesetzt ist.
E249:  F0 20     BEQ  $E26B       Ja, kein 64er Modul vorhanden

******************************    Kernal Routine: GO64MODE
                                  System als C-64 konfigurieren

E24B:  A9 E3     LDA  # $E3       C-64 System Werte in
E24D:  85 01     STA  * $01       Datenregister Prozessorport
E24F:  A9 2F     LDA  # $2F       C-64 System Werte in
E251:  85 00     STA  * $00       Datenrichtungsregister Prozessorport
E253:  A2 08     LDX  # $08       8 Bytes sind zu kopieren
```

```
E255:  BD 62 E2   LDA  $E262,X     Hier wird das ROM Original der
E258:  95 01      STA  * $01,X     Routine, die den C-64 konfiguriert,
E25A:  CA         DEX              in die Z-Page kopiert, da diese
E25B:  D0 F8      BNE  $E255       Routine nur dort ablaufen darf.
E25D:  8E 30 D0   STX  $D030       Taktfrequenz auf 1MHz setzen
E260:  4C 02 00   JMP  $0002       Zur Z-Page Routine: Konfig. C-64

******************************     Diese Routine dient dazu, den C-128
                                   als C-64 zu konfigurieren. Sie kann
                                   nur in der Z-Page ablaufen, da alle
                                   anderen Bereiche abgeschaltet werden

E263:  A9 F7      LDA  # $F7       Initialisierungswert für C-64 System
E265:  8D 05 D5   STA  $D505       in das MCR Register d. MMU schreiben
E268:  6C FC FF   JMP  ($FFFC)     RESET Vektor C-64 anspringen

******************************     Funktion-ROM Test im C-128 Mode

E26B:  A2 03      LDX  # $03       Schleifen- und Displacementzähler
E26D:  8E C0 0A   STX  $0AC0       für Modultest initialisieren
E270:  A9 00      LDA  # $00       Die ersten 4 Bytes der PAT
E272:  9D C1 0A   STA  $0AC1,X     (Physikalische Adreß-Tabelle der
E275:  CA         DEX              Erweiterungskarten) werden hier
E276:  10 FA      BPL  $E272       gelöscht. ($00 initialisiert)
E278:  85 9E      STA  * $9E       Lo Adreßwert für Modultest
E27A:  A0 09      LDY  # $09       Displacement auf Modulcode (CBM)
E27C:  AE C0 0A   LDX  $0AC0       Displacementzähler für Modulcheck
E27F:  BD BC E2   LDA  $E2BC,X     Holt Hi Adreßwert aus Tabelle
E282:  85 9F      STA  * $9F       und legt ihn in Z-Page ab
E284:  BD C0 E2   LDA  $E2C0,X     Holt Bank Wert für Test aus Tabelle
E287:  85 02      STA  * $02       und legt ihn in Z-Page Bank Byte ab
E289:  A6 02      LDX  * $02       Hole Bank Code aus Z-Page
E28B:  A9 9E      LDA  # $9E       Hole Adresse $9E als VETVEC in Akku
E28D:  20 D0 F7   JSR  $F7D0       Rout. INDFET: LDA(fetvec),Y bel.Bank
E290:  D9 BD E2   CMP  $E2BD,Y     1 Zeichen auf "CBM" Code testen
E293:  D0 21      BNE  $E2B6       Nicht gleich, nächste Bank/Adresse
E295:  88         DEY              Weiter auf "CBM" Code prüfen
E296:  C0 07      CPY  # $07       Wenn die 3 Codezeichen erkannt sind,
E298:  B0 EF      BCS  $E289       weiter, sonst in Testschleife
E29A:  A6 02      LDX  * $02       Hole Bank Code des aktuellen Tests
E29C:  A9 9E      LDA  # $9E       Hole Adresse $9E als FETVEC in Akku
E29E:  20 D0 F7   JSR  $F7D0       Rout. INDFET: LDA(fetvec),Y bel.Bank
E2A1:  AE C0 0A   LDX  $0AC0       Hole F-ROM Displacement Zeiger
E2A4:  9D C1 0A   STA  $0AC1,X     Prüfe ID-Tab. der Erweiterungskarten
E2A7:  C9 01      CMP  # $01       Prüfe, ob Erweiterung angemeldet ist
E2A9:  D0 0B      BNE  $E2B6       Nein, dann Skip zum nächsten Test
E2AB:  A5 9E      LDA  * $9E       Lo der Einsprungadresse in Akku
E2AD:  A4 9F      LDY  * $9F       Hi der Einsprungadresse in Y-Reg
```

```
E2AF:  85 04      STA  * $04     Lo der Einsprungadresse in PC-Lo
E2B1:  84 03      STY  * $03     Hi der Einsprungadresse in PC-Hi
E2B3:  20 CD 02   JSR  $02CD     JSRFAR Rout.: JSR beliebig.Bank +RTS
E2B6:  CE C0 0A   DEC  $0AC0     Schleifen-/Displacementzähler -1
E2B9:  10 BF      BPL  $E27A     Nicht Null, dann weiter testen
E2BB:  60         RTS            Rücksprung aus dem Unterprogramm

******************************   Hi Adressen für Modulprüfung

E2BC:  C0 80 C0 80               $C000, $8000, $C000, $8000

******************************   Bank Nummer für Modulprüfung

E2C0:  04 04 08 08               Int.ROM, Int.ROM, Ext.ROM, Ext.ROM

******************************   Code zur Modulkennzeichnung

E2C4:  43 42 4D                  <C> <B> <M>

******************************   Initialisierungstabelle für VIC Reg.

E2C7:  00 00 00 00 00 00 00 00
E2CF:  00 00 00 00 00 00 00 00
E2D7:  00 1B FF 00 00 00 08 00
E2DF:  14 FF 01 00 00 00 00 00
E2E7:  0D 0B 01 02 03 01 02 00
E2EF:  01 02 03 04 05 06 07 FF
E2F7:  FC

******************************   Initialisierungstabelle für VDC Reg.

E2F8:  00 7E 01 50 02 66 03 49   VDC-Tab 1
E300:  04 20 05 00 06 19 07 1D
E308:  08 00 09 07 0A 20 0B 07
E310:  0C 00 0D 00 0E 00 0F 00
E318:  14 08 15 00 17 08 18 20
E320:  19 40 1A F0 1B 00 1C 20
E328:  1D 07 22 7D 23 64 24 05
E330:  16 78
E332:  FF         .Byte  $FF     Trennzeichen
E333:  19 47                     VDC-Tab 2
E335:  FF         .Byte  $FF     Trennzeichen
E336:  04 27 07 20               VDC-Tab 3
E33A:  FF         .Byte  $FF     Trennzeichen

******************************   Kernal Routine: TALK

E33B:  09 40      ORA  # $40     Bit 6 für TALK setzen
```

```
E33D:  2C        .Byte  $2C     Skip nach $E340
E33E:  09 20     ORA  # $20     Bit 5 für LISTN setzen
E340:  20 EC E7  JSR   $E7EC    Ende der RS-232 Übertragung abwarten

******************************  Kernal Routine: LISTN

E343:  48        PHA            TALK/LISTN KZ auf Stapel sichern
E344:  24 94     BIT  * $94     Ist weiteres Byte auszugeben?
E346:  10 0A     BPL   $E352    Nein, dann weiter
E348:  38        SEC            Carry für Rotation setzen
E349:  66 A3     ROR  * $A3     Flag für EOI setzen
E34B:  20 8C E3  JSR   $E38C    Byte auf IEC Bus ausgeben
E34E:  46 94     LSR  * $94     Zeichen im Puffer KZ löschen
E350:  46 A3     LSR  * $A3     Flag für EOI wieder löschen
E352:  68        PLA            Alten Akku Inhalt zurückholen
E353:  85 95     STA  * $95     auszugebendes Byte in Z-Page sichern
E355:  20 73 E5  JSR   $E573    SEI, 1 MHz, Sprites abschalten
E358:  20 57 E5  JSR   $E557    Data  Hi ausgeben
E35B:  AD 00 DD  LDA   $DD00    Prüfe, ob am Datenport A des CIA 2
E35E:  29 08     AND  # $08     das ATN Signal gesetzt ist
E360:  D0 12     BNE   $E374    Nicht gesetzt, dann Skip
E362:  20 D6 E5  JSR   $E5D6    Impuls für schnellen seriellen Modus
E365:  A9 FF     LDA  # $FF     Ein-/Ausgabe Datenpuffer f. serielle
E367:  8D 0C DC  STA   $DC0C    Übertragung auf High-Value setzen
E36A:  20 BC E5  JSR   $E5BC    Rückmeldung vom Bus abwarten
E36D:  8A        TXA            X-Reg Inhalt in Akku sichern
E36E:  A2 14     LDX  # $14     Schleifenzähler auf 20 setzen
E370:  CA        DEX            Schleifenzähler um 1 vermindern
E371:  D0 FD     BNE   $E370    Warten, bis Schleife heruntergezählt
E373:  AA        TAX            Alten X-Reg Inhalt wiederherstellen
E374:  AD 00 DD  LDA   $DD00    Port A des CIA 2 auslesen
E377:  09 08     ORA  # $08     ATN Lo Signal setzen und in den
E379:  8D 00 DD  STA   $DD00    Port A des CIA 2 zurückschreiben
E37C:  20 73 E5  JSR   $E573    Taktfrequenz 1MHz,Sprites abschalten
E37F:  20 4E E5  JSR   $E54E    Clock Lo ausgeben
E382:  20 57 E5  JSR   $E557    Data  Hi ausgeben

******************************  Verzögerungsschleife ca. 1 Millisek.

E385:  8A        TXA            X-Reg Inhalt im Akku sichern
E386:  A2 B8     LDX  # $B8     Schleifenzähler auf 184 setzen
E388:  CA        DEX            Schleifenzähler um 1 vermindern
E389:  D0 FD     BNE   $E388    Schleifen, bis Zähler = Null ist
E38B:  AA        TAX            X-Reg Inhalt wiederherstellen

******************************  Byte auf IEC Bus (vorbereiten)

E38C:  20 73 E5  JSR   $E573    Taktfrequenz 1MHz,Sprites abschalten
```

```
E38F:  20 57 E5   JSR  $E557      Data  Hi ausgeben
E392:  20 69 E5   JSR  $E569      Bit vom IEC Bus ins Carry holen
E395:  90 03      BCC  $E39A      Data nicht Lo, dann Ok und Skip
E397:  4C 28 E4   JMP  $E428      "Device not present"in System Status
E39A:  2C 0D DC   BIT  $DC0D      Teste CIA Interrupt Steuerregister
E39D:  20 45 E5   JSR  $E545      Clock Hi ausgeben
E3A0:  24 A3      BIT  * $A3      Z-Page Zeiger für EOI gesetzt?
E3A2:  10 0A      BPL  $E3AE      Nein, dann Skip
E3A4:  20 69 E5   JSR  $E569      Bit vom IEC Bus ins Carry holen
E3A7:  90 FB      BCC  $E3A4      Warten auf Data  Lo Signal
E3A9:  20 69 E5   JSR  $E569      Bit vom IEC Bus ins Carry holen
E3AC:  B0 FB      BCS  $E3A9      Warten auf Data  Hi Signal
E3AE:  AD 00 DD   LDA  $DD00      Hier werden die Daten vom Port A
E3B1:  CD 00 DD   CMP  $DD00      des CIA 2 gelesen und die gelesenen
E3B4:  D0 F8      BNE  $E3AE      Daten entprellt
E3B6:  48         PHA             Gelesene Daten auf Stack sichern
E3B7:  AD 0D DC   LDA  $DC0D      Prüfe Interrupt Steuerregister
E3BA:  29 08      AND  # $08      Ist Timer A auf "One Shot"?
E3BC:  F0 05      BEQ  $E3C3      Ja, dann Skip
E3BE:  A9 C0      LDA  # $C0      Kontrollbit 6 und 7 im System-Zeiger
E3C0:  8D 1C 0A   STA  $0A1C      für schnellen seriellen Modus setzen
E3C3:  68         PLA             Gelesene Daten vom Stack zurückholen
E3C4:  10 E8      BPL  $E3AE      Bit 7 gelöscht, dann Skip
E3C6:  09 10      ORA  # $10      Bit 4 für Clock Ausgang an ser. Bus
E3C8:  8D 00 DD   STA  $DD00      setzen und in Port A schreiben
E3CB:  29 08      AND  # $08      Prüfe, ob Bit 3 gesetzt ist
E3CD:  D0 13      BNE  $E3E2      Nein, dann Skip
E3CF:  2C 1C 0A   BIT  $0A1C      Prüfe Bit 7 des ser. Mode Zeigers
E3D2:  10 0E      BPL  $E3E2      Bit 7 gelöscht, dann Skip
E3D4:  20 D6 E5   JSR  $E5D6      Impuls für schnellen seriellen Modus
E3D7:  A5 95      LDA  * $95      Gesichertes Byte zurückholen und in
E3D9:  8D 0C DC   STA  $DC0C      CIA Ein-/Ausgabe Register schreiben
E3DC:  20 BC E5   JSR  $E5BC      Rückmeldung vom Bus abwarten
E3DF:  4C 12 E4   JMP  $E412      Byteausgabe über seriellen Bus

*******************************   Byte auf IEC Bus (ausgeben)

E3E2:  A9 08      LDA  # $08      Zähler für die Anzahl der zu
E3E4:  85 A5      STA  * $A5      sendenden Bits mit 8 initialisieren
E3E6:  AD 00 DD   LDA  $DD00      Hier werden die Daten vom Port A
E3E9:  CD 00 DD   CMP  $DD00      des CIA 2 gelesen und die gelesenen
E3EC:  D0 F8      BNE  $E3E6      Daten noch entprellt.
E3EE:  0A         ASL  A          Data in das Carry Flag schieben
E3EF:  90 34      BCC  $E425      Data Hi ausgeben, "time out" ausgeben
E3F1:  66 95      ROR  * $95      Bit zur Ausgabe bereitstellen
E3F3:  B0 05      BCS  $E3FA      Prüfe, ob Bit gesetzt ist
E3F5:  20 60 E5   JSR  $E560      Nein, dann Data Lo ausgeben
E3F8:  D0 03      BNE  $E3FD      und zur Clock Hi Ausgabe springen
```

```
E3FA:  20 57 E5  JSR  $E557      Data Hi ausgeben
E3FD:  20 45 E5  JSR  $E545      Clock Hi ausgeben
E400:  EA        NOP             Leerschritt: No Operation
E401:  EA        NOP             Leerschritt: No Operation
E402:  EA        NOP             Leerschritt: No Operation
E403:  EA        NOP             Leerschritt: No Operation
E404:  AD 00 DD  LDA  $DD00      Port A des CIA 2 auslesen
E407:  29 DF     AND  # $DF      Bit 5: Data Ausgang ser. Bus löschen
E409:  09 10     ORA  # $10      Bit 4: Clock Ausgang ser. Bus setzen
E40B:  8D 00 DD  STA  $DD00      in den Datenport A zurückschreiben
E40E:  C6 A5     DEC  * $A5      Bitzähler um 1 vermindern. Wenn
E410:  D0 D4     BNE  $E3E6      weiteres Bit ausgeben,dann schleifen
E412:  8A        TXA             Inhalt des X-Reg in Akku kopieren
E413:  48        PHA             und X-Reg auf Stack sichern
E414:  A2 22     LDX  # $22      Hi Impuls Zähler auf #34 setzen
E416:  20 69 E5  JSR  $E569      Ein Bit vom IEC Bus ins Carry holen
E419:  B0 05     BCS  $E420      Data Hi, dann Skip
E41B:  68        PLA             Alten X-Reg Inhalt vom Stack holen
E41C:  AA        TAX             und X-Reg Inhalt wiederherstellen
E41D:  4C 9F E5  JMP  $E59F      Taktfrequenz + Sprites zurücksetzen
E420:  CA        DEX             Data Hi Zähler um 1 vermindern
E421:  D0 F3     BNE  $E416      Noch keine 22 Hi Impulse,dann weiter
E423:  68        PLA             Alten X-Reg Inhalt vom Stack holen
E424:  AA        TAX             und X-Reg Inhalt wiederherstellen
E425:  A9 03     LDA  # $03      Code für System Status: Time out
E427:  2C        .Byte  $2C      Skip nach $E42A
E428:  A9 80     LDA  # $80      Code für Status: Device not present
E42A:  48        PHA             Status Code auf Stapel sichern
E42B:  AD 1C 0A  LDA  $0A1C      Prüfe den Fast seriell Mode Zeiger
E42E:  29 7F     AND  # $7F      Bit 7 ausblenden, nur fast/slow
E430:  8D 1C 0A  STA  $0A1C      In Fast Mode Flag zurückschreiben
E433:  68        PLA             Status Fehler Code zurückholen
E434:  20 57 F7  JSR  $F757      System Status neu setzen
E437:  20 9F E5  JSR  $E59F      Taktfrequenz + Sprites zurücksetzen
E43A:  18        CLC             Kennzeichen für OK setzen
E43B:  4C 35 E5  JMP  $E535      Gerät über UNLSN Routine abhängen

********************************  Kernal Routine: ACPTR
                                 Byte vom seriellen Bus holen

E43E:  20 73 E5  JSR  $E573      Systemtakt 1MHz, Sprites abschalten
E441:  A9 00     LDA  # $00      Den Z-Page Zeiger für die serielle
E443:  85 A5     STA  * $A5      EOI Kennzeichnung löschen
E445:  2C 0D DC  BIT  $DC0D      Bit 7 des CIA-ISR auslesen
E448:  8A        TXA             Den aktuellen Inhalt des X-Reg über
E449:  48        PHA             den Akku auf dem Stapel sichern
E44A:  20 45 E5  JSR  $E545      Clock Hi Signal an Port A
E44D:  20 69 E5  JSR  $E569      Bit vom IEC Bus ins Carry holen
```

```
E450:  10 FB     BPL  $E44D     Warten auf DATA Hi Signal
E452:  A2 0D     LDX  # $0D     Schleifenzähler mit #13 initial.
E454:  AD 00 DD  LDA  $DD00     Datenport A des CIA 2 auslesen
E457:  29 DF     AND  # $DF     Bit 6: "Ser. Bus Impuls ein" löschen
E459:  8D 00 DD  STA  $DD00     und in Datenport zurückschreiben
E45C:  AD 00 DD  LDA  $DD00     Datenport A des CIA 2 lesen und
E45F:  CD 00 DD  CMP  $DD00     warten, bis ein Bit über den Bus
E462:  D0 F8     BNE  $E45C     am Port angekommen ist
E464:  0A        ASL  A         Datenbit in das Carry Flag schieben
E465:  10 1D     BPL  $E484     Datenbyte vom Bus holen
E467:  CA        DEX            Schleifenzähler um 1 vermindern
E468:  D0 F2     BNE  $E45C     Schleife nicht Null, dann Skip
E46A:  A5 A5     LDA  * $A5     Prüfe Z-Page EOI Zeiger
E46C:  D0 0F     BNE  $E47D     Bei #0, EOI empfangen, sonst Skip
E46E:  20 60 E5  JSR  $E560     Data Lo Signal an seriellen Bus
E471:  20 45 E5  JSR  $E545     Clock Hi Signal an seriellen Bus
E474:  A9 40     LDA  # $40     Code für Status: EOI-Leitung
E476:  20 57 F7  JSR  $F757     System Status neu setzen
E479:  E6 A5     INC  * $A5     EOI Zeiger auf Zeitfehler b. Timeout
E47B:  D0 D5     BNE  $E452     Datenbyte nach EOI holen
E47D:  68        PLA            Den gesicherten X-Reg Inhalt über
E47E:  AA        TAX            den Akku vom Stack zurückholen
E47F:  A9 02     LDA  # $02     Code für Status: Timeout b. Lesen
E481:  4C 2A E4  JMP  $E42A     System Status neu setzen
E484:  A2 08     LDX  # $08     Zähler für 8 Datenbits setzen
E486:  AD 0D DC  LDA  $DC0D     Interrupt Steuerregister auslesen
E489:  29 08     AND  # $08     Prüfe, auf Timer, Clock oder Bus
E48B:  D0 28     BNE  $E4B5     Interrupt. Ja, dann Skip
E48D:  AD 00 DD  LDA  $DD00     Datenport A des CIA 2 lesen und
E490:  CD 00 DD  CMP  $DD00     warten, bis ein Bit über den Bus
E493:  D0 F8     BNE  $E48D     am Port angekommen ist
E495:  0A        ASL  A         Datenbit in das Carry Flag schieben
E496:  10 EE     BPL  $E486     Nein, warten bis Daten gültig sind
E498:  66 A4     ROR  * $A4     Empfangenes Datenbit in Bitspeicher
E49A:  AD 00 DD  LDA  $DD00     Datenport A des CIA 2 lesen und
E49D:  CD 00 DD  CMP  $DD00     warten, bis ein Bit über den Bus
E4A0:  D0 F8     BNE  $E49A     am Port angekommen ist
E4A2:  0A        ASL  A         Datenbit in das Carry Flag schieben
E4A3:  30 F5     BMI  $E49A     Nein, dann warten
E4A5:  CA        DEX            Zähler für die Datenbitzahl -1
E4A6:  F0 17     BEQ  $E4BF     8 Datenbits angekommen, dann Skip
E4A8:  AD 00 DD  LDA  $DD00     Datenport A des CIA 2 lesen und
E4AB:  CD 00 DD  CMP  $DD00     warten, bis ein Bit über den Bus
E4AE:  D0 F8     BNE  $E4A8     am Port angekommen ist
E4B0:  0A        ASL  A         Datenbit in das Carry Flag schieben
E4B1:  10 F5     BPL  $E4A8     Sprung, wenn empfangenes Bit "0"
E4B3:  30 E3     BMI  $E498     Sprung, wenn empfangenes Bit "1"
E4B5:  AD 0C DC  LDA  $DC0C     Inhalt des Ein-/Ausgabe Datenpuffers
```

```
E4B8:  85 A4     STA  * $A4    in der Z-Page zwischenspeichern
E4BA:  A9 C0     LDA  # $C0    Bit 6 und 7 im System Flag für den
E4BC:  8D 1C 0A  STA  $0A1C    Fast seriell Mode setzen
E4BF:  68        PLA           Den gesicherten X-Reg Inhalt über
E4C0:  AA        TAX           den Akku vom Stapel zurückholen
E4C1:  20 60 E5  JSR  $E560    Data Lo Signal an seriellen Bus
E4C4:  24 90     BIT  * $90    STATUS auf gesetztes EOF Bit testen
E4C6:  50 03     BVC  $E4CB    Kein EOF gefunden, dann weiter
E4C8:  20 38 E5  JSR  $E538    Gerät über UNLSN Routine abhängen
E4CB:  20 9F E5  JSR  $E59F    Taktfrequenz + Sprites zurücksetzen
E4CE:  A5 A4     LDA  * $A4    Datenbyte in den Akku holen
E4D0:  18        CLC           Kennzeichen für OK setzen
E4D1:  60        RTS           Rücksprung aus dem Unterprogramm

******************************  Kernal Routine: SECND
                                Sekundäradresse nach LISTN senden

E4D2:  85 95     STA  * $95    Sekundäradresse in Z-Page sichern
E4D4:  20 7C E3  JSR  $E37C    mit Attention (ATN) ausgeben
E4D7:  AD 00 DD  LDA  $DD00    Datenport A des CIA 2 lesen
E4DA:  29 F7     AND  # $F7    Bit 3 ausblenden und so das ATN
E4DC:  8D 00 DD  STA  $DD00    Signal an ser. Bus zurücknehmen
E4DF:  60        RTS           Rücksprung aus dem Unterprogramm

******************************  Kernal Routine: TKSA

E4E0:  85 95     STA  * $95    Sekundäradresse in Z-Page sichern
E4E2:  20 7C E3  JSR  $E37C    mit Attention (ATN) ausgeben
E4E5:  24 90     BIT  * $90    STATUS auf gesetztes EOF Bit testen
E4E7:  30 4C     BMI  $E535    EOF aufgetreten, dann in UNLSN Rout.
E4E9:  20 73 E5  JSR  $E573    Taktfrequenz 1MHz,Sprites abschalten
E4EC:  20 60 E5  JSR  $E560    Data Lo Signal an seriellen Bus
E4EF:  20 D7 E4  JSR  $E4D7    Einsprung in SECND Routine
E4F2:  20 45 E5  JSR  $E545    Clock Hi Siganl an ser. Bus
E4F5:  AD 00 DD  LDA  $DD00    Datenport A des CIA 2 lesen und
E4F8:  CD 00 DD  CMP  $DD00    warten, bis ein Bit über den Bus
E4FB:  D0 F8     BNE  $E4F5    am Port angekommen ist
E4FD:  0A        ASL  A        Datenbit ins Carry Flag schieben
E4FE:  30 F5     BMI  $E4F5    und auf Data Hi warten
E500:  4C 9F E5  JMP  $E59F    Taktfrequenz + Sprites zurücksetzen

******************************  Kernal Routine: CIOUT

E503:  24 94     BIT  * $94    Ist noch weiteres Byte auszugeben?
E505:  30 05     BMI  $E50C    Ja, dann in Ausgabeschleife
E507:  38        SEC           Carry für Rotation auf "1" setzen
E508:  66 94     ROR  * $94    Flag für gepuffertes Byte setzen
E50A:  D0 05     BNE  $E511    Ausgabeschleife überspringen
```

```
E50C:  48        PHA           Das Byte auf den Stack retten
E50D:  20 8C E3  JSR  $E38C    Gepuffertes Byte auf Bus ausgeben
E510:  68        PLA           Byte vom Stack zurückholen
E511:  85 95     STA  * $95    Im Z-Page Ausgabespeicher ablegen
E513:  18        CLC           Carry für "OK" KZ setzen
E514:  60        RTS           Rücksprung aus dem Unterprogramm

*****************************  Kernal Routine: UNTLK

E515:  20 73 E5  JSR  $E573    Taktfrequenz zurücksetzen
E518:  20 4E E5  JSR  $E54E    Clock Lo Signal an Port A
E51B:  AD 00 DD  LDA  $DD00    Datenport A des CIA 2 lesen
E51E:  09 08     ORA  # $08    Bit 3 in diesen Wert einblenden und
E520:  8D 00 DD  STA  $DD00    ATN LO Signal auf Bus geben
E523:  A9 5F     LDA  # $5F    Code für UNTLK in Akku laden
E525:  2C        .Byte  $2C    Skip nach $E528

*****************************  Kernal Routine: UNLSN

E526:  A9 3F     LDA  # $3F    Code für UNLSN in Akku laden
E528:  48        PHA           und auf dem Stapel zwischenspeichern
E529:  AD 1C 0A  LDA  $0A1C    Status Zeiger für "Fast Seriell"
E52C:  29 7F     AND  # $7F    Bit 7 ausblenden
E52E:  8D 1C 0A  STA  $0A1C    und wieder zurückschreiben
E531:  68        PLA           Alten Akku Inhalt wiederherstellen
E532:  20 43 E3  JSR  $E343    Kernal Routine: LISTN
E535:  20 D7 E4  JSR  $E4D7    ATN zurücksetzen, Hi
E538:  8A        TXA           X-Reg Inhalt im Akku sichern
E539:  A2 0A     LDX  # $0A    Zeitschleife für 40 Mikrosekunden
E53B:  CA        DEX           Schleifenzähler um 1 vermindern
E53C:  D0 FD     BNE  $E53B    Warten bis Schleife abgearbeitet
E53E:  AA        TAX           Alten X-Reg Inhalt wiederherstellen
E53F:  20 45 E5  JSR  $E545    Clock Hi Signal an Port A
E542:  4C 57 E5  JMP  $E557    Data  Hi Signal an Port A und RTS

*****************************  Clock Hi Signal

E545:  AD 00 DD  LDA  $DD00    Datenport A des CIA 2 lesen
E548:  29 EF     AND  # $EF    Bit 4 für Clock Ausgang an ser. Bus
E54A:  8D 00 DD  STA  $DD00    löschen und in Port A schreiben
E54D:  60        RTS           Rücksprung aus dem Unterprogramm

*****************************  Clock Lo Signal

E54E:  AD 00 DD  LDA  $DD00    Datenport A des CIA 2 lesen
E551:  09 10     ORA  # $10    Bit 4 für Clock Ausgang an ser. Bus
E553:  8D 00 DD  STA  $DD00    setzen und in Port A schreiben
```

```
E556:  60          RTS          Rücksprung aus dem Unterprogramm

******************************  Data Hi Signal

E557:  AD 00 DD    LDA  $DD00   Datenport A des CIA 2 lesen
E55A:  29 DF       AND  # $DF   Bit 5 für Data Ausgang an ser. Bus
E55C:  8D 00 DD    STA  $DD00   löschen und in Port A schreiben
E55F:  60          RTS          Rücksprung aus dem Unterprogramm

******************************  Data Lo Signal

E560:  AD 00 DD    LDA  $DD00   Datenport A des CIA 2 lesen
E563:  09 20       ORA  # $20   Bit 5 für Data Ausgang an ser. Bus
E565:  8D 00 DD    STA  $DD00   setzen und in Port A schreiben
E568:  60          RTS          Rücksprung aus dem Unterprogramm

******************************  Ein Bit vom IEC Bus ins Carry holen

E569:  AD 00 DD    LDA  $DD00   Datenport A des CIA 2 lesen und
E56C:  CD 00 DD    CMP  $DD00   warten, bis ein Bit über den Bus
E56F:  D0 F8       BNE  $E569   am Port angekommen ist.
E571:  0A          ASL  A       Empfangenes Bit (Bit 7) ins Carry
E572:  60          RTS          Rücksprung aus dem Unterprogramm

******************************  Systemtaktfrequenz auf 1MHz setzen
                                und alle Sprites abschalten

E573:  78          SEI          Alle System Interrupts verhindern
E574:  2C 3A 0A    BIT  $0A3A   Prüfe Interrupt Sicherungsspeicher
E577:  30 25       BMI  $E59E   Bit 7 gesetzt, dann Rücksprung
E579:  2C 37 0A    BIT  $0A37   Prüfe Taktfreq. Sicherungsspeicher
E57C:  30 20       BMI  $E59E   Bit 7 gesetzt, dann Rücksprung
E57E:  AD 30 D0    LDA  $D030   VIC Register für Taktfrequenz
E581:  8D 37 0A    STA  $0A37   in Systemspeicher sichern
E584:  AD 15 D0    LDA  $D015   VIC Register für Sprite enable
E587:  8D 38 0A    STA  $0A38   in Systemspeicher sichern
E58A:  A9 00       LDA  # $00   Init. Status f. 1 MHz, keine Sprites
E58C:  8D 15 D0    STA  $D015   Alle Sprites abschalten
E58F:  8D 30 D0    STA  $D030   Taktfrequenz auf 1 MHz setzen
E592:  AD 38 0A    LDA  $0A38   Waren Sprites eingeschaltet?
E595:  F0 07       BEQ  $E59E   Nein, dann Rücksprung
E597:  8A          TXA          X-Reg Inhalt im Akku sichern
E598:  A2 00       LDX  # $00   Warteschleife für 1.3 Millisekunden
E59A:  CA          DEX          Schleifenzähler um 1 vermindern
E59B:  D0 FD       BNE  $E59A   Warteschleife ganz abarbeiten
E59D:  AA          TAX          Alten X-Reg Inhalt zurückholen
E59E:  60          RTS          Rücksprung aus dem Unterprogramm
```

```
******************************   Taktfrequenz + Spritezeiger auf den
                                 ursprünglichen Status zurücksetzen

E59F:  2C 3A 0A   BIT  $0A3A     Prüfe Interrupt Sicherungsspeicher
E5A2:  30 16      BMI  $E5BA     Bit 7 gesetzt, dann Rücksprung
E5A4:  2C 37 0A   BIT  $0A37     Prüfe Taktfreq. Sicherungsspeicher
E5A7:  10 11      BPL  $E5BA     Frequenz nicht geändert, dann Skip
E5A9:  AD 38 0A   LDA  $0A38     Den gesicherten Wert des Sprite-
E5AC:  8D 15 D0   STA  $D015     Enable Registers zurückschreiben
E5AF:  AD 37 0A   LDA  $0A37     Den gesicherten Wert für die
E5B2:  8D 30 D0   STA  $D030     System Taktfrequenz zurückschreiben
E5B5:  A9 00      LDA  # $00     Zwischenspeicher für Sicherung der
E5B7:  8D 37 0A   STA  $0A37     System-Taktfrequenz löschen
E5BA:  58         CLI            Alle System Interrupts freigeben
E5BB:  60         RTS            Rücksprung aus dem Unterprogramm

******************************   Rückmeldung vom Bus abwarten

E5BC:  AD 0D DC   LDA  $DC0D     Hole CIA Interrupt Steuerregister
E5BF:  29 08      AND  # $08     Warte, bis Bit 4 (SRQ Eingabe vom
E5C1:  F0 F9      BEQ  $E5BC     seriellen Bus) gelöscht ist.
E5C3:  AD 0E DC   LDA  $DC0E     Steuerregister A des CIA auslesen
E5C6:  29 80      AND  # $80     Bit 7 f. 50 Hz Frequenz eliminieren
E5C8:  09 08      ORA  # $08     Timer auf Modus Toggle und
E5CA:  8D 0E DC   STA  $DC0E     "One Shot" setzen und Timer starten
E5CD:  AD 05 D5   LDA  $D505     Das Kontrollbit für den Fast-Seriell
E5D0:  29 F7      AND  # $F7     Mode im Mode Configuration Register
E5D2:  8D 05 D5   STA  $D505     der MMU ausblenden
E5D5:  60         RTS            Rücksprung aus dem Unterprogramm

******************************   Fast Impuls an seriellen Bus

E5D6:  AD 05 D5   LDA  $D505     Das Kontrollbit für den Fast seriell
E5D9:  09 08      ORA  # $08     Mode im Mode Configuration Register
E5DB:  8D 05 D5   STA  $D505     der MMU setzen
E5DE:  A9 7F      LDA  # $7F     Code für Interrupts löschen
E5E0:  8D 0D DC   STA  $DC0D     an Interrupt Steuerregister
E5E3:  A9 00      LDA  # $00     Den Timer A Hi im CIA 2 mit dem
E5E5:  8D 05 DC   STA  $DC05     High-Wert #0 vorladen
E5E8:  A9 04      LDA  # $04     Den Timer A Lo im CIA 2 mit dem
E5EA:  8D 04 DC   STA  $DC04     Low-Wert #4 vorladen
E5ED:  AD 0E DC   LDA  $DC0E     Steuerregister A des CIA auslesen
E5F0:  29 80      AND  # $80     Bit 7 f. 50 Hz Frequenz eliminieren
E5F2:  09 55      ORA  # $55     Timer auf Modus Force Load, Toggle,
E5F4:  8D 0E DC   STA  $DC0E     ser. Bus aus und Timer A starten
E5F7:  2C 0D DC   BIT  $DC0D     Interrupt Steuerregister auslesen
E5FA:  60         RTS            Rücksprung aus dem Unterprogramm
```

```
*****************************   Kernal Routine: FSTMOD

E5FB:  90 C6     BCC  $E5C3    Rückmeldung vom ser. Bus abwarten
E5FD:  B0 D7     BCS  $E5D6    Fast Impuls an seriellen Bus

*****************************   RS-232 Ausgabe

E5FF:  A5 B4     LDA  * $B4    Anzahl der zu sendenden Bits
E601:  F0 47     BEQ  $E64A    Ist Byte komplett übertragen?
E603:  30 3F     BMI  $E644    Ist Stopbit erforderlich?
E605:  46 B6     LSR  * $B6    Nächstes Bit ins Carry Flag schieben
E607:  A2 00     LDX  # $00    X-Reg als KZ mit $00 initialisieren
E609:  90 01     BCC  $E60C    Bit gelöscht?
E60B:  CA        DEX           Nein, dann X-Reg auf $FF setzen
E60C:  8A        TXA           Bit gelöscht, KZ in Akku kopieren,
E60D:  45 BD     EOR  * $BD    mit Parity Status verknüpfen und
E60F:  85 BD     STA  * $BD    wieder in Z-Page Parity abspeichern
E611:  C6 B4     DEC  * $B4    Bitzähler um 1 vermindern
E613:  F0 06     BEQ  $E61B    Alle Bits übertragen, dann weiter
E615:  8A        TXA           X-Reg Inhalt in Akku kopieren
E616:  29 04     AND  # $04    Bit 2 isolieren
E618:  85 B5     STA  * $B5    und in Ausgaberegister bringen
E61A:  60        RTS           Rücksprung aus dem Unterprogramm

*****************************   Sendeparität überprüfen

E61B:  A9 20     LDA  # $20    Bit 5 für Parity in Akku einblenden
E61D:  2C 11 0A  BIT  $0A11    Prüfe RS-232 Befehlsregister
E620:  F0 14     BEQ  $E636    Betriebsmodus ohne Parity, dann Skip
E622:  30 1C     BMI  $E640    Betriebsmodus mit fester Parität?
E624:  70 14     BVS  $E63A    Betriebsmodus für ungerade Parität?
E626:  A5 BD     LDA  * $BD    Parität gleich eins?
E628:  D0 01     BNE  $E62B    Nein, dann Skip
E62A:  CA        DEX           Parität auf $FFsetzen
E62B:  C6 B4     DEC  * $B4    Bitzähler auf $FF setzen
E62D:  AD 10 0A  LDA  $0A10    Hole RS-232 Kontrollregister in Akku
E630:  10 E3     BPL  $E615    Sind zwei Stopbits erforderlich?
E632:  C6 B4     DEC  * $B4    Bitzähler auf $FE setzen
E634:  D0 DF     BNE  $E615    Nicht Null, Stopbits berechnen
E636:  E6 B4     INC  * $B4    Bitzähler +1, keine Parität
E638:  D0 F0     BNE  $E62A    Nicht Null, Stopbits berechnen
E63A:  A5 BD     LDA  * $BD    Hole Parity Wert aus Z-Page
E63C:  F0 ED     BEQ  $E62B    Bei Null ein 0-Bit ausgeben
E63E:  D0 EA     BNE  $E62A    Nicht Null, dann ein 1-Bit ausgeben
E640:  70 E9     BVS  $E62B    Routine: 0-Bit ausgeben
E642:  50 E6     BVC  $E62A    Routine: 1-Bit ausgeben (Parity fix)
E644:  E6 B4     INC  * $B4    Bitzähler um 1 erhöhen
E646:  A2 FF     LDX  # $FF    Codewert f. Stopbit in X-Reg bringen
```

```
E648:  D0 CB     BNE  $E615      Unbedingter Sprung

*****************************    3-Line / X-Line Handshake Test

E64A:  AD 11 0A  LDA  $0A11      Lade Akku mit RS-232 Befehlsregister
E64D:  4A        LSR  A          Bit 0 in das Carry Flag schieben
E64E:  90 07     BCC  $E657      3-Line Handshake Abfrage übergehen
E650:  2C 01 DD  BIT  $DD01      Port B des CIA 2 auslesen
E653:  10 1D     BPL  $E672      Fehlt DATA SET READY (DSR) Signal?
E655:  50 1E     BVC  $E675      Fehlt CLEAR TO SEND  (CTS) Signal?
E657:  A9 00     LDA  # $00      Den Z-Page Puffer für die RS-232
E659:  85 BD     STA  * $BD      Parität löschen ($00) und den Z-Page
E65B:  85 B5     STA  * $B5      Speicher für zu sendendes Startbit
E65D:  AE 15 0A  LDX  $0A15      Anzahl der zu übertragenden Bits
E660:  86 B4     STX  * $B4      als Zähler in Z-Page kopieren
E662:  AC 1A 0A  LDY  $0A1A      Index auf Anfang Ausgabepuffer mit
E665:  CC 1B 0A  CPY  $0A1B      Ende vergleichen. Wenn alle Bytes
E668:  F0 13     BEQ  $E67D      übertragen, dann abschließen
E66A:  B1 CA     LDA  ($CA),Y    Datenbyte aus RS-232 Puffer holen
E66C:  85 B6     STA  * $B6      und in Sendespeicher übergeben
E66E:  EE 1A 0A  INC  $0A1A      Index: Anfang Ausgabepuffer erhöhen
E671:  60        RTS             Rücksprung aus dem Unterprogramm

*****************************    NMI Status für RS-232 setzen

E672:  A9 40     LDA  # $40      Code für DATA SET READY (DSR) fehlt
E674:  2C        .Byte  $2C      Skip nach $E677
E675:  A9 10     LDA  # $10      Code für CLEAR TO SEND (CTS) fehlt
E677:  0D 14 0A  ORA  $0A14      Mit RS-232 Statusregister verknüpfen
E67A:  8D 14 0A  STA  $0A14      und im Statusregister ablegen
E67D:  A9 01     LDA  # $01      Akku mit $01 laden und den
E67F:  8D 0D DD  STA  $DD0D      NMI für den Timer A löschen
E682:  4D 0F 0A  EOR  $0A0F      Mit RS-232 NMI Status verknüpfen
E685:  09 80     ORA  # $80      Flag für RS-232 umdrehen und diesen
E687:  8D 0F 0A  STA  $0A0F      Wert im RS-232 NMI Status ablegen
E68A:  8D 0D DD  STA  $DD0D      Alle weiteren NMI's zulassen
E68D:  60        RTS             Rücksprung aus dem Unterprogramm

*****************************    Anzahl d. RS-232 Datenbits berechnen

E68E:  A2 09     LDX  # $09      Defaultwert auf 8 Datenbits
E690:  A9 20     LDA  # $20      Prüfwert für Anzahl der Datenbits
E692:  2C 10 0A  BIT  $0A10      RS-232 Kontrollregister überprüfen
E695:  F0 01     BEQ  $E698      Bit 5 gelöscht?
E697:  CA        DEX             Datenbitzahl um 1 vermindern
E698:  50 02     BVC  $E69C      Bit 6 gelöscht?
E69A:  CA        DEX             Datenbitzahl um 1 vermindern
E69B:  CA        DEX             Datenbitzahl um 1 vermindern
```

```
E69C:  60          RTS          Rücksprung aus dem Unterprogramm

*****************************   Empfangenes Bit verarbeiten

E69D:  A6 A9       LDX  * $A9   Prüfe, ob es sich dabei um
E69F:  D0 33       BNE  $E6D4   ein Startbit handelt. Nein, Skip
E6A1:  C6 A8       DEC  * $A8   Bitzähler um 1 vermindern
E6A3:  F0 3A       BEQ  $E6DF   Alle Bits empfangen, dann weiter
E6A5:  30 0D       BMI  $E6B4   Wenn Stopbits zu erwarten, dann Skip
E6A7:  A5 A7       LDA  * $A7   Empfangenes Bit in Akku holen
E6A9:  45 AB       EOR  * $AB   und für Parität verknüpfen
E6AB:  85 AB       STA  * $AB   Paritätswert in Z-Page ablegen
E6AD:  46 A7       LSR  * $A7   Empfangenes Bit ins Carry Flag und
E6AF:  66 AA       ROR  * $AA   in Eingabebyte Puffer schieben
E6B1:  60          RTS          Rücksprung aus dem Unterprogramm

*****************************   Startbitzeiger setzen, wenn alle
                                Stopbits empfangen wurden

E6B2:  C6 A8       DEC  * $A8   Bitzähler um 1 vermindern
E6B4:  A5 A7       LDA  * $A7   Stopbit-Wert in Akku holen und
E6B6:  F0 6B       BEQ  $E723   prüfen, ob dieser Null ist. Skip
E6B8:  AD 10 0A    LDA  $0A10   RS-232 Kontrollregister in Akku
E6BB:  0A          ASL  A       Anzahl der Stopbits ins Carry
E6BC:  A9 01       LDA  # $01   Additionswert für Stopbitzahl
E6BE:  65 A8       ADC  * $A8   Datenbits und Stopbits addieren
E6C0:  D0 EF       BNE  $E6B1   Nicht alle Stopbits empfangen, Skip
E6C2:  A9 90       LDA  # $90   RXD über Flag empfangen in Akku
E6C4:  8D 0D DD    STA  $DD0D   und den NMI damit freigeben
E6C7:  0D 0F 0A    ORA  $0A0F   Mit RS-232 NMI Status verknüpfen
E6CA:  8D 0F 0A    STA  $0A0F   und im RS-232 NMI Status ablegen
E6CD:  85 A9       STA  * $A9   Flag für Startbit setzen
E6CF:  A9 02       LDA  # $02   Akku für Übergabe mit 2 init.
E6D1:  4C 7F E6    JMP  $E67F   und den NMI für Timer B löschen

*****************************   Prüfe auf RS-232 Startbit

E6D4:  A5 A7       LDA  * $A7   Startbitwert in Akku holen
E6D6:  D0 EA       BNE  $E6C2   Nicht Null, dann Skip. Sonst das
E6D8:  85 A9       STA  * $A9   Z-Page Startbit Flag zurücksetzen
E6DA:  A9 01       LDA  # $01   und den Z-Page Zeiger für RS-232
E6DC:  85 AB       STA  * $AB   Eingabeparität zurücksetzen
E6DE:  60          RTS          Rücksprung aus dem Unterprogramm

*****************************   Empfangenes Byte verarbeiten

E6DF:  AC 18 0A    LDY  $0A18   Den Index auf den Anfang des RS-232
E6E2:  C8          INY          Eingabepuffers um 1 erhöhen
```

```
E6E3:  CC 19 0A   CPY  $0A19       Mit Ende vergleichen. Wenn Puffer,
E6E6:  F0 2A      BEQ  $E712       dann entsprechend Status setzen
E6E8:  8C 18 0A   STY  $0A18       Pufferindex zurückschreiben
E6EB:  88         DEY              und wieder um 1 vermindern
E6EC:  A5 AA      LDA  * $AA       Empfangenes Byte aus Z-Page holen
E6EE:  AE 15 0A   LDX  $0A15       Anzahl der Datenbits in X-Reg
E6F1:  E0 09      CPX  # $09       8 Bits, 1 Stopbit empfangen?
E6F3:  F0 04      BEQ  $E6F9       Ja, dann alles OK
E6F5:  4A         LSR  A           Bits in richtige Position schieben
E6F6:  E8         INX              Datenbitzähler um 1 erhöhen
E6F7:  D0 F8      BNE  $E6F1       Sprung zur Byte-Justierung
E6F9:  91 C8      STA  ($C8),Y     Byte in Eingabepuffer schreiben
E6FB:  A9 20      LDA  # $20       Kontrollwert für Paritätsprüfung
E6FD:  2C 11 0A   BIT  $0A11       RS-232 Kommandoregister prüfen
E700:  F0 B0      BEQ  $E6B2       Übertragung ist ohne Parität
E702:  30 AD      BMI  $E6B1       Fester Bitwert für Parität
E704:  A5 A7      LDA  * $A7       Empfangenes Paritätsbit in Akku
E706:  45 AB      EOR  * $AB       mit errechneter Parität vergleichen
E708:  F0 03      BEQ  $E70D       Gleich, dann weiter mit OK
E70A:  70 A5      BVS  $E6B1       Gerade Parität, dann weiter mit OK
E70C:  2C         .Byte  $2C       Skip nach $E70F
E70D:  50 A2      BVC  $E6B1       Ungerade Parität, dann weiter mit OK
E70F:  A9 01      LDA  # $01       Code für Paritätsfehler in Akku
E711:  2C         .Byte  $2C       Skip nach $E714
E712:  A9 04      LDA  # $04       Empfangspuffer voll Code in Akku
E714:  2C         .Byte  $2C       Skip nach $E717
E715:  A9 80      LDA  # $80       Break Befehl empfangen in Akku
E717:  2C         .Byte  $2C       Skip nach $E71A
E718:  A9 02      LDA  # $02       Rahmen-Fehler Code in Akku
E71A:  0D 14 0A   ORA  $0A14       Code mit RS-232 Status verknüpfen
E71D:  8D 14 0A   STA  $0A14       und im RS-232 Statusregister ablegen
E720:  4C C2 E6   JMP  $E6C2       Sprung: Empfang des nächsten Bytes

******************************     RS-232 CKOUT, Ausgabe auf RS-232

E723:  A5 AA      LDA  * $AA       Empfangenes Byte in Akku holen
E725:  D0 F1      BNE  $E718       Rahmen-Fehler
E727:  F0 EC      BEQ  $E715       Break Befehl empfangen
E729:  85 9A      STA  * $9A       Gerätenummer in Z-Page ablegen
E72B:  AD 11 0A   LDA  $0A11       RS-232 Kommandoregister laden
E72E:  4A         LSR  A           Bit 0 (Handshake) ins Carry schieben
E72F:  90 29      BCC  $E75A       Sprung bei 3-Line Handshake
E731:  A9 02      LDA  # $02       Code für DATA SET READY Test in Akku
E733:  2C 01 DD   BIT  $DD01       Port B des CIA 2 auslesen
E736:  10 1D      BPL  $E755       Kein DSR Signal, dann Fehler
E738:  D0 20      BNE  $E75A       Kein Request to Send Signal
E73A:  AD 0F 0A   LDA  $0A0F       RS-232 NMI Status in Akku holen
E73D:  29 02      AND  # $02       Wenn Datenempfang aktiv ist, dann
```

```
E73F:  D0 F9     BNE  $E73A      warten, bis Empfang beendet ist.
E741:  2C 01 DD  BIT  $DD01      Port B des CIA 2 auslesen und auf
E744:  70 FB     BVS  $E741      Clear to Send Signal warten
E746:  AD 01 DD  LDA  $DD01      Port B des CIA 2 auslesen und Bit 2
E749:  09 02     ORA  # $02      f. Request to Send Signal einblenden
E74B:  8D 01 DD  STA  $DD01      In Port B zurückschreiben
E74E:  2C 01 DD  BIT  $DD01      Port B des CIA2 auslesen und auf
E751:  70 07     BVS  $E75A      Clear to Send Signal warten
E753:  30 F9     BMI  $E74E      Data Set Ready abfragen
E755:  A9 40     LDA  # $40      Code für fehlendes Data Set Ready
E757:  8D 14 0A  STA  $0A14      Signal in RS-232 Status schreiben
E75A:  18        CLC             Carry für Kennzeichen OK setzen
E75B:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   Ausgabe in RS-232 Puffer
                                 CTS = Clear to send
                                 DSR = Data set ready

E75C:  20 70 E7  JSR  $E770      Falls notwendig, Übertragung starten
E75F:  AC 1B 0A  LDY  $0A1B      Index auf Ende RS-232 Ausgabepuffer
E762:  C8        INY             in Y-Reg holen und um 1 erhöhen
E763:  CC 1A 0A  CPY  $0A1A      Mit Index Anfang Ausgabepuffer vgl.
E766:  F0 F4     BEQ  $E75C      Puffer voll, dann warten
E768:  8C 1B 0A  STY  $0A1B      Neuen Index auf Ausgabepuffer setzen
E76B:  88        DEY             und diesen Zeiger um 1 vermindern
E76C:  A5 9E     LDA  * $9E      Auszugebendes Byte in Akku holen
E76E:  91 CA     STA  ($CA),Y    und in Ausgabepuffer schreiben
E770:  AD 0F 0A  LDA  $0A0F      RS-232 NMI Flag in Akku kopieren
E773:  4A        LSR  A          Teste, ob Bit 0 gesetzt ist
E774:  B0 1E     BCS  $E794      Läuft der Sendebetrieb schon?
E776:  A9 10     LDA  # $10      Den Timer A mit $10 initialisieren
E778:  8D 0E DD  STA  $DD0E      und anschließend starten
E77B:  AD 16 0A  LDA  $0A16      Den 2 Byte Timer für die
E77E:  8D 04 DD  STA  $DD04      Sende Baud-Rate in
E781:  AD 17 0A  LDA  $0A17      $DD04-$DD05
E784:  8D 05 DD  STA  $DD05      neu setzen
E787:  A9 81     LDA  # $81      Code für Timer A Unterlaufs NMI
E789:  20 7F E6  JSR  $E67F      NMI bei Unterlauf von Timer A
E78C:  20 4A E6  JSR  $E64A      CTS+DSR prüfen,Übertragung freigeben
E78F:  A9 11     LDA  # $11      Den Timer A mit $11 initialisieren
E791:  8D 0E DD  STA  $DD0E      und anschließend starten
E794:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   RS-232 CHKIN, RS-232 Eingabe setzen

E795:  85 99     STA  * $99      Gerätenummer in Z-Page ablegen
E797:  AD 11 0A  LDA  $0A11      RS-232 Kommandoregister in Akku
E79A:  4A        LSR  A          Bit 0 (Handshake) ins Carry schieben
```

```
E79B:  90 28     BCC  $E7C5      3-Line Handshake, dann weiter
E79D:  29 08     AND  # $08      Duplex Betrieb testen
E79F:  F0 24     BEQ  $E7C5      Voll Duplex, dann weiter
E7A1:  A9 02     LDA  # $02      Code für DSR Signal Test
E7A3:  2C 01 DD  BIT  $DD01      Port B des CIA 2 abfragen auf DSR
E7A6:  10 AD     BPL  $E755      Fehlt, dann Status setzen und Exit
E7A8:  F0 22     BEQ  $E7CC      Ready to Send Signal abfragen
E7AA:  AD 0F 0A  LDA  $0A0F      RS-232 NMI Status Flag in Akku
E7AD:  4A        LSR  A          Wenn Sendebetrieb aktiv ist, dann
E7AE:  B0 FA     BCS  $E7AA      warten bis Übertragung beendet
E7B0:  AD 01 DD  LDA  $DD01      Port B des CIA 2 auslesen und
E7B3:  29 FD     AND  # $FD      Bit 0 f. Request to Send eliminieren
E7B5:  8D 01 DD  STA  $DD01      Signal an Port B zurückgeben
E7B8:  AD 01 DD  LDA  $DD01      Port B des CIA 2 auslesen und
E7BB:  29 04     AND  # $04      DATA TERMINAL READY Signal prüfen
E7BD:  F0 F9     BEQ  $E7B8      Nicht vorhanden, dann warten
E7BF:  A9 90     LDA  # $90      NMI Maske für "Flag" in Akku holen
E7C1:  18        CLC             Carry als OK Kennzeichen löschen
E7C2:  4C 7F E6  JMP  $E67F      RS-232 NMI freigeben

******************************   RS-232 CHKIN bei 3-Line Handshake

E7C5:  AD 0F 0A  LDA  $0A0F      RS-232 NMI Status in Akku holen
E7C8:  29 12     AND  # $12      Wenn die RS-232 noch nicht aktiv
E7CA:  F0 F3     BEQ  $E7BF      ist, dann starten
E7CC:  18        CLC             Carry als OK Kennzeichen löschen
E7CD:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   GET von RS-232

E7CE:  AD 14 0A  LDA  $0A14      RS-232 Status Byte in Akku holen
E7D1:  AC 19 0A  LDY  $0A19      Index auf Ende RS-232 Eingabepuffers
E7D4:  CC 18 0A  CPY  $0A18      Mit Index Anfang Eingabepuffer vgl.
E7D7:  F0 0B     BEQ  $E7E4      Wenn gleich, dann Puffer leer: Skip
E7D9:  29 F7     AND  # $F7      Bit 3 (Puffer leer) ausblenden
E7DB:  8D 14 0A  STA  $0A14      und im RS-232 Status löschen
E7DE:  B1 C8     LDA  ($C8),Y    Ein Byte aus RS-232 Puffer lesen
E7E0:  EE 19 0A  INC  $0A19      Index auf RS-232 Eingabepuffer + 1
E7E3:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   GET von RS-232 wenn Puffer leer

E7E4:  09 08     ORA  # $08      Das Bit 3 (Kennzeichen für Puffer
E7E6:  8D 14 0A  STA  $0A14      leer) im RS-232 Status einblenden
E7E9:  A9 00     LDA  # $00      $00 als gelesenes Zeichen übergeben
E7EB:  60        RTS             Rücksprung aus dem Unterprogramm
```

```
******************************        Ende der RS-232 Übertragung abwarten

E7EC:  48          PHA                Akku Inhalt auf Stack retten
E7ED:  AD 0F 0A    LDA  $0A0F         RS-232 NMI Flag holen
E7F0:  F0 11       BEQ  $E803         Nicht gesetzt, dann OK und weiter
E7F2:  AD 0F 0A    LDA  $0A0F         RS-232 NMI Flag erneut lesen
E7F5:  29 03       AND  # *03         Bit 0 = senden, Bit 1 = empfangen
E7F7:  D0 F9       BNE  $E7F2         Ende abwarten
E7F9:  A9 10       LDA  # $10         Akku mit $10 laden
E7FB:  8D 0D DD    STA  $DD0D         Interrupt über "Flag" Leitung
E7FE:  A9 00       LDA  # $00         RS-232 NMI Flag
E800:  8D 0F 0A    STA  $0A0F         auf Status "OK" setzen
E803:  68          PLA                Alten Akku Inhalt zurückholen
E804:  60          RTS                Rücksprung aus dem Unterprogramm


******************************        NMI Routine für RS-232

E805:  98          TYA                Interrupt Control Register (ICR)
E806:  2D 0F 0A    AND  $0A0F         mit RS-232 NMI Flag verknüpfen
E809:  AA          TAX                und Ergebnis im X-Reg sichern
E80A:  29 01       AND  # $01         Bits 1-7 ausblenden und prüfen, ob
E80C:  F0 28       BEQ  $E836         Sendebetrieb aktiv ist. Nein: Skip
E80E:  AD 00 DD    LDA  $DD00         Akku mit Datenport laden
E811:  29 FB       AND  # $FB         Bit 2 (TXD) löschen und das zu
E813:  05 B5       ORA  * $B5         sendende Bit übergeben
E815:  8D 00 DD    STA  $DD00         Anschließend im Datenport speichern
E818:  AD 0F 0A    LDA  $0A0F         RS-232 NMI Flag in den Akku kopieren
E81B:  8D 0D DD    STA  $DD0D         und wieder in das ICR schreiben
E81E:  8A          TXA                ICR/RS-232 NMI Verknüpfung in Akku
E81F:  29 12       AND  # $12         Die Bits 1 und 4 isolieren
E821:  F0 0D       BEQ  $E830         Nicht gesetzt, Byteempfang einleiten
E823:  29 02       AND  # $02         Bit 1, Aufruf von Timer B isolieren
E825:  F0 06       BEQ  $E82D         Nicht gesetzt, dann Startbit
E827:  20 78 E8    JSR  $E878         Empfangenes Bit weiterverarbeiten
E82A:  4C 30 E8    JMP  $E830         Empfang eines Bytes einleiten
E82D:  20 A9 E8    JSR  $E8A9         Vorbereitung f.Empfang nächstes Byte
E830:  20 FF E5    JSR  $E5FF         Empfang des Bytes einleiten
E833:  4C 49 E8    JMP  $E849         Rückkehr zum Interrupt
E836:  8A          TXA                X-Reg Inhalt im Akku sichern
E837:  29 02       AND  # $02         Datenempfang?
E839:  F0 06       BEQ  $E841         Nein, dann Skip Verarbeitung
E83B:  20 78 E8    JSR  $E878         Empfangenes Bit verarbeiten
E83E:  4C 49 E8    JMP  $E849         Rückkehr vom Interrupt
E841:  8A          TXA                Alten X-Reg Inhalt wiederherstellen
E842:  29 10       AND  # $10         Prüfe, ob auf ein Startbit gewartet
E844:  F0 03       BEQ  $E849         werden muß. Nein, dann weiter
E846:  20 A9 E8    JSR  $E8A9         Nächsten Bitempfang vorbereiten
```

```
E849:  AD 0F 0A   LDA  $0A0F     RS-232 NMI Flag laden
E84C:  8D 0D DD   STA  $DD0D     und in ICR des CIA 2 kopieren
E84F:  60         RTS            Rücksprung aus dem Unterprogramm

*****************************    Timerkonstanten für RS-232 Baudrate
                                 Tabelle 1 für NTSC Version

E850:  C1 27      (= 10177)        50    Baud
E852:  3E 1A      (=  6718)        75    Baud
E854:  C5 11      (=  4549)       110    Baud
E856:  74 0E      (=  3700)       134,5  Baud
E858:  ED 0C      (=  3309)       150    Baud
E85A:  45 06      (=  1605)       300    Baud
E85C:  F0 02      (=   752)       600    Baud
E85E:  46 01      (=   326)      1200    Baud
E860:  B8 00      (=   184)      1800    Baud
E862:  71 00      (=   113)      2400    Baud

*****************************    Timerkonstanten für RS-232 Baudrate
                                 Tabelle 2 für PAL Version

E864:  19 26      (=  9753)        50    Baud
E866:  44 19      (=  6468)        75    Baud
E868:  1A 11      (=  4378)       110    Baud
E86A:  E8 0D      (=  3560)       134.5  Baud
E86C:  70 0C      (=  3184)       150    Baud
E86E:  06 06      (=  1542)       300    Baud
E870:  D1 02      (=   736)       600    Baud
E872:  37 01      (=   311)      1200    Baud
E874:  AE 00      (=   174)      1800    Baud
E876:  69 00      (=   105)      2400    Baud

*****************************    NMI Routine für RS-232 Eingabe

E878:  AD 01 DD   LDA  $DD01     Datenport B des CIA 2 auslesen
E87B:  29 01      AND  # $01     Bit für Receive Data isolieren
E87D:  85 A7      STA  * $A7     und in Z-Page RS-232 Eingabebitflag
E87F:  AD 06 DD   LDA  $DD06     Lo Wert des CIA 2 Timers B holen
E882:  E9 28      SBC  # $28     und davon 28 subtrahieren
E884:  6D 16 0A   ADC  $0A16     Full-Bit-Time Baudrate Lo addieren
E887:  8D 06 DD   STA  $DD06     und Timer B neu setzen
E88A:  AD 07 DD   LDA  $DD07     Hi Wert des CIA 2 Timers B holen
E88D:  6D 17 0A   ADC  $0A17     Full-Bit-Time Baudrate Hi addieren
E890:  8D 07 DD   STA  $DD07     und Timer B Hi neu setzen
E893:  A9 11      LDA  # $11     $11 in das Kontrollregister des
E895:  8D 0F DD   STA  $DD0F     CIA 2 schreiben = Timer B starten
E898:  AD 0F 0A   LDA  $0A0F     RS-232 NMI Status in Akku holen
E89B:  8D 0D DD   STA  $DD0D     und CIA Interrupt-Steuerreg. setzen
```

```
E89E:  A9 FF     LDA  # $FF      Initialisierungswert für Timer B
E8A0:  8D 06 DD  STA  $DD06      Timer B Lo auf High Value setzen
E8A3:  8D 07 DD  STA  $DD07      Timer B Hi auf High Value setzen
E8A6:  4C 9D E6  JMP  $E69D      Empfangenes Bit verarbeiten

*******************************  NMI Routine für RS-232 Ausgabe

E8A9:  AD 12 0A  LDA  $0A12      RS-232 Benutzerbaudrate in Akku
E8AC:  8D 06 DD  STA  $DD06      und in Timer B Lo des CIA 2
E8AF:  AD 13 0A  LDA  $0A13      RS-232 Benutzerbaudrate in Akku
E8B2:  8D 07 DD  STA  $DD07      und in Timer B Hi des CIA 2
E8B5:  A9 11     LDA  # $11      $11 in das Kontrollregister des
E8B7:  8D 0F DD  STA  $DD0F      CIA 2 schreiben = Timer starten
E8BA:  A9 12     LDA  # $12      Bits 0, 1 und 4 des RS-232 NMI
E8BC:  4D 0F 0A  EOR  $0A0F      Flags invertieren. Diesen Wert
E8BF:  8D 0F 0A  STA  $0A0F      zurück in das NMI Flag
E8C2:  A9 FF     LDA  # $FF      Initialisierungswert für Timer B
E8C4:  8D 06 DD  STA  $DD06      Timer B Lo auf High Value setzen
E8C7:  8D 07 DD  STA  $DD07      Timer B Hi auf High Value setzen
E8CA:  AE 15 0A  LDX  $0A15      Anzahl der zu sendenden Bits
E8CD:  86 A8     STX  * $A8      in Z-Page: Zähler für RS-232 Bits
E8CF:  60        RTS             Rücksprung aus dem Unterprogramm

*******************************  Programm Header vom Band lesen

E8D0:  A5 93     LDA  * $93      Load/Verify Zeiger über den Akku
E8D2:  48        PHA             auf dem Systemstack retten
E8D3:  20 F2 E9  JSR  $E9F2      Routine: Datenblock v. Band lesen
E8D6:  68        PLA             Das Load/Verify Flag vom Stack
E8D7:  85 93     STA  * $93      zurückholen und wieder in Z-Page
E8D9:  B0 3D     BCS  $E918      Wenn Fehler aufgetreten, Rücksprung
E8DB:  A0 00     LDY  # $00      Displacement auf Bandpuffer setzen
E8DD:  B1 B2     LDA  ($B2),Y    Hole Byte des gelesenen Headerblocks
E8DF:  C9 05     CMP  # $05      War es das EOT Kennzeichen?
E8E1:  F0 34     BEQ  $E917      Ja, dann Rücksprung
E8E3:  C9 01     CMP  # $01      Header-Typ für Basic-Programm?
E8E5:  F0 08     BEQ  $E8EF      Ja, dann entsprechend auswerten
E8E7:  C9 03     CMP  # $03      Header-Typ für Maschinenprogramm?
E8E9:  F0 04     BEQ  $E8EF      Ja, dann entsprechend auswerten
E8EB:  C9 04     CMP  # $04      Header Typ für Datenblock?
E8ED:  D0 E1     BNE  $E8D0      Nein, dann neu einlesen
E8EF:  AA        TAX             Header Typ in X-Reg sichern
E8F0:  24 9D     BIT  * $9D      Prüfe Kernal Nachrichten Flag
E8F2:  10 22     BPL  $E916      Steuermeldungen nicht erlaubt, Skip
E8F4:  A0 63     LDY  # $63      Displacement auf "FOUND" Meldung
E8F6:  20 22 F7  JSR  $F722      Steuermeldung ausgeben
E8F9:  A0 05     LDY  # $05      Displ. auf Dateinamenanfang setzen
E8FB:  B1 B2     LDA  ($B2),Y    Zeichen aus Bandpuffer lesen
```

```
E8FD:  20 D2 FF   JSR  $FFD2      Kernal BSOUT: Ein Zeichen ausgeben
E900:  C8         INY             Displacementzeiger um 1 erhöhen
E901:  C0 15      CPY  # $15      Maximale Dateinamenlänge =16 Zeichen
E903:  D0 F6      BNE  $E8FB      Noch nicht erreicht, dann weiter
E905:  A5 A1      LDA  * $A1      Mittelwertiges Time Byte in Akku
E907:  69 02      ADC  # $02      Warteschleife für 8,5 Sekunden
E909:  A4 91      LDY  * $91      Prüfe Z-Page Stop / C= Tasten Flag
E90B:  C8         INY             erhöhe diesen Wert um 1
E90C:  D0 04      BNE  $E912      Taste gedrückt, dann weiter
E90E:  C5 A1      CMP  * $A1      Prüfe die 8.5 Sekunden Warteschleife
E910:  D0 F7      BNE  $E909      Zeit nicht um, dann weiter warten
E912:  C0 F0      CPY  # $F0      Wurde die Space Taste gedrückt?
E914:  F0 BA      BEQ  $E8D0      Ja, dann Header einlesen
E916:  18         CLC             Kennzeichen für alles OK setzen
E917:  88         DEY             Alten Stop / C= Tasten Flagwert
E918:  60         RTS             Rücksprung aus dem Unterprogramm

******************************    Datenblock auf Band schreiben
                                  Header auf Band schreiben, Headertyp
                                  im Akku: 3 = Maschinenspr.,1 = Basic

E919:  85 9E      STA  * $9E      Headertyp in Z-Page ablegen
E91B:  20 80 E9   JSR  $E980      Hole Bandpufferadresse aus Z-Page
E91E:  90 5F      BCC  $E97F      Adresse ungültig, dann Skip
E920:  A5 C2      LDA  * $C2      Startadresse Hi in den Akku bringen
E922:  48         PHA             und auf dem Stack sichern
E923:  A5 C1      LDA  * $C1      Startadresse Lo in den Akku bringen
E925:  48         PHA             und auf dem Stack sichern
E926:  A5 AF      LDA  * $AF      Endadresse Hi in den Akku bringen
E928:  48         PHA             und auf dem Stack sichern
E929:  A5 AE      LDA  * $AE      Endadresse Lo in den Akku bringen
E92B:  48         PHA             und auf dem Stack sichern
E92C:  A0 BF      LDY  # $BF      Bandpufferlänge für Schleife holen
E92E:  A9 20      LDA  # $20      Akku mit Zeichen für <Blank> laden
E930:  91 B2      STA  ($B2),Y    Bandpuffer löschen
E932:  88         DEY             Schleifen, bis die ganze in Y-Reg
E933:  D0 FB      BNE  $E930      übergebene Länge gelöscht ist
E935:  A5 9E      LDA  * $9E      Hole den gesicherten Header-Typ
E937:  91 B2      STA  ($B2),Y    an 1. Position im Bandpuffer
E939:  C8         INY             Displacement auf Bandpuffer + 1
E93A:  A5 C1      LDA  * $C1      Hole Startadresse Lo aus Z-Page
E93C:  91 B2      STA  ($B2),Y    und bringe sie in den Bandpuffer
E93E:  C8         INY             Displacement auf Bandpuffer + 1
E93F:  A5 C2      LDA  * $C2      Hole Startadresse Hi aus Z-Page
E941:  91 B2      STA  ($B2),Y    und bringe sie in den Bandpuffer
E943:  C8         INY             Displacement auf Bandpuffer + 1
E944:  A5 AE      LDA  * $AE      Hole Endadresse Lo aus Z-Page
E946:  91 B2      STA  ($B2),Y    und bringe sie in den Bandpuffer
```

```
E948:  C8        INY            Displacement auf Bandpuffer + 1
E949:  A5 AF     LDA  * $AF     Hole Endadresse Hi aus Z-Page
E94B:  91 B2     STA  ($B2),Y   und bringe sie in den Bandpuffer
E94D:  C8        INY            Displacement auf Bandpuffer + 1
E94E:  84 9F     STY  * $9F     Displ. im Bandpuffer merken
E950:  A0 00     LDY  # $00     Zähler für Länge des Dateinamens
E952:  84 9E     STY  * $9E     in Z-Page löschen
E954:  A4 9E     LDY  * $9E     Zähler für Dateinamenlänge holen
E956:  C4 B7     CPY  * $B7     und mit echter Länge vergleichen
E958:  F0 0D     BEQ  $E967     Alle Buchstaben im Puffer, dann Skip
E95A:  20 AE F7  JSR  $F7AE     Buchstaben aus Dateinamens holen
E95D:  A4 9F     LDY  * $9F     Displ. auf Bandpuffer zurückholen
E95F:  91 B2     STA  ($B2),Y   Buchstabe des Dateinamen in Puffer
E961:  E6 9E     INC  * $9E     Zähler für Dateinamenlänge + 1
E963:  E6 9F     INC  * $9F     Displacement auf Bandpuffer + 1
E965:  D0 ED     BNE  $E954     Schleifen für nächsten Buchstaben
E967:  20 87 E9  JSR  $E987     Start- und Endadresse auf Bandpuffer
E96A:  A9 69     LDA  # $69     Checksumme für Header- Datenblock
E96C:  85 AB     STA  * $AB     ($69) in Z-Page zwischenspeichern
E96E:  20 1C EA  JSR  $EA1C     Block auf Band schreiben
E971:  A8        TAY            Aktuellen Akku Inhalt retten
E972:  68        PLA            Endadresse Hi vom Stack zurückholen
E973:  85 AE     STA  * $AE     und wieder in Z-Page ablegen
E975:  68        PLA            Endadresse Lo vom Stack zurückholen
E976:  85 AF     STA  * $AF     und wieder in Z-Page ablegen
E978:  68        PLA            Startadresse Hi v. Stack zurückholen
E979:  85 C1     STA  * $C1     und wieder in Z-Page ablegen
E97B:  68        PLA            Startadresse Lo v. Stack zurückholen
E97C:  85 C2     STA  * $C2     und wieder in Z-Page ablegen
E97E:  98        TYA            Gesicherten Akku Inhalt zurückholen
E97F:  60        RTS            Rücksprung aus dem Unterprogramm

******************************  Hole Bandpuffer Adresse und prüfe
                                auf Gültigkeit

E980:  A6 B2     LDX  * $B2     Anfang Bandpuffer Lo in X-Reg
E982:  A4 B3     LDY  * $B3     Anfang Bandpuffer Hi in Y-Reg
E984:  C0 02     CPY  # $02     Zero-Page und Stack nicht erlaubt
E986:  60        RTS            Rücksprung aus dem Unterprogramm

******************************  Bandendadresse = Startadresse + 192

E987:  20 80 E9  JSR  $E980     Hole Bandpuffer Adresse
E98A:  8A        TXA            Anfang Bandpuffer Lo in Akku
E98B:  85 C1     STA  * $C1     und in Z-Page E-/A Startadresse Lo
E98D:  18        CLC            Carry für Addition löschen
E98E:  69 C0     ADC  # $C0     Endadresse = Startadresse + 192
E990:  85 AE     STA  * $AE     Neue Endadresse Lo in Z-Page
```

```
E992:  98         TYA             Anfang Bandpuffer Hi in Akku
E993:  85 C2      STA   * $C2     und in Z-Page E-/A Startadresse Hi
E995:  69 00      ADC   # $00     Endadresse Hi = Startadresse Hi +
E997:  85 AF      STA   * $AF     Überlauf, Endadresse Hi in Z-Page
E999:  60         RTS             Rücksprung aus dem Unterprogramm

******************************    Bandheader nach Namen suchen

E99A:  20 D0 E8   JSR   $E8D0     Nächsten Bandheader suchen
E99D:  B0 1E      BCS   $E9BD     Wenn EOT gefunden, dann Rücksprung
E99F:  A0 05      LDY   # $05     Displacement auf Namen in Kas-Puffer
E9A1:  84 9F      STY   * $9F     in Z-Page Zwischenspeicher bringen
E9A3:  A0 00      LDY   # $00     Zähler für die Länge des Dateinamens
E9A5:  84 9E      STY   * $9E     in der Zero-Page initialisieren
E9A7:  C4 B7      CPY   * $B7     Mit Länge Suchnamen vergleichen
E9A9:  F0 11      BEQ   $E9BC     Wenn gleich, dann weiter auswerten
E9AB:  20 AE F7   JSR   $F7AE     Zeichen des "Such-Dateinamens" holen
E9AE:  A4 9F      LDY   * $9F     Displ. auf Dateinamen im Bandpuffer
E9B0:  D1 B2      CMP   ($B2),Y   Mit Suchzeichen vergleichen
E9B2:  D0 E6      BNE   $E99A     Nicht gleich, dann nicht gefunden
E9B4:  E6 9E      INC   * $9E     Dateinamenlängenzähler +1
E9B6:  E6 9F      INC   * $9F     Dateinamendispl. auf Kas-Buffer +1
E9B8:  A4 9E      LDY   * $9E     Dateinamenlängenzähler in Y-Reg
E9BA:  D0 EB      BNE   $E9A7     Nächster Zeichenvergleich
E9BC:  18         CLC             Kennzeichen für OK setzen
E9BD:  60         RTS             Rücksprung aus dem Unterprogramm

******************************    Bandpuffer-Zeiger erhöhen

E9BE:  20 80 E9   JSR   $E980     Hole die Bandpufferadresse
E9C1:  E6 A6      INC   * $A6     Z-Page Kassettenpufferzeiger +1
E9C3:  A4 A6      LDY   * $A6     und anschließend auf den
E9C5:  C0 C0      CPY   # $C0     Maximalwert 192 vergleichen
E9C7:  60         RTS             Rücksprung aus dem Unterprogramm

******************************    Wartet auf Taste an der Datassette

E9C8:  20 DF E9   JSR   $E9DF     Prüfen, ob Taste gedrückt
E9CB:  F0 1A      BEQ   $E9E7     Taste gedrückt, dann OK und weiter
E9CD:  A0 1B      LDY   # $1B     Displ. auf "Press Play on Tape" in Y
E9CF:  20 22 F7   JSR   $F722     Steuermeldung ausgeben
E9D2:  20 8F EA   JSR   $EA8F     Teste auf Stop Tasten Unterbrechung
E9D5:  20 DF E9   JSR   $E9DF     Prüfen, ob Taste gedrückt
E9D8:  D0 F8      BNE   $E9D2     Nein, dann in Warteschleife
E9DA:  A0 6A      LDY   # $6A     Displacement für "OK" Meldung
E9DC:  4C 22 F7   JMP   $F722     Steuermeldung ausgeben
```

```
******************************     Prüfen, ob Band Taste gedrückt ist

E9DF:  A9 10     LDA  # $10    Für Tastentest Bit 4 setzen
E9E1:  24 01     BIT  * $01    Prüfe Datenregister Prozessorport
E9E3:  D0 02     BNE  $E9E7    Nicht gedrückt, dann Exit
E9E5:  24 01     BIT  * $01    Erneut prüfen (Entprellung)
E9E7:  18        CLC           Ja: Zero-Flag = 1, Nein Z-Flag = 0
E9E8:  60        RTS           Rücksprung aus dem Unterprogramm

******************************     Warten auf "Record & Play" Taste

E9E9:  20 DF E9  JSR  $E9DF    Prüfen, ob Band Taste gedrückt ist
E9EC:  F0 F9     BEQ  $E9E7    Taste gedrückt, dann OK und weiter
E9EE:  A0 2E     LDY  # $2E    Displ. auf "Press R & P on Tape"
E9F0:  D0 DD     BNE  $E9CF    Tasten Warteschleife / Stop Abfrage

******************************     Datenblock vom Band einlesen

E9F2:  A9 00     LDA  # $00    Systemstatus mit Kennzeichen
E9F4:  85 90     STA  * $90    für alles OK initialisieren
E9F6:  85 93     STA  * $93    Load/Verify Zeiger löschen
E9F8:  20 87 E9  JSR  $E987    Bandpufferadresse/Endadresse holen

******************************     Programm vom Band laden

E9FB:  20 C8 E9  JSR  $E9C8    Wartet auf Taste an Datassette
E9FE:  B0 1F     BCS  $EA1F    STOP-Taste gedrückt, Rücksprung
EA00:  78        SEI           Alle System Interrupts verhindern
EA01:  A9 00     LDA  # $00    Initialisierungswert f. IRQ-Speicher
EA03:  85 AA     STA  * $AA    Band Read Modus Eingabebyte Speicher
EA05:  85 B4     STA  * $B4    Band Hilfszeiger
EA07:  85 B0     STA  * $B0    Kassetten Zeitkonstante
EA09:  85 9E     STA  * $9E    Kassettenfehler Pass 1
EA0B:  85 9F     STA  * $9F    Kassettenfehler Pass 2
EA0D:  85 9C     STA  * $9C    Band Flag für Byte empfangen
EA0F:  A9 90     LDA  # $90    IRQ an Pin "Flag"
EA11:  A2 0E     LDX  # $0E    Nummer des IRQ Vektors ($EAEB)
EA13:  D0 11     BNE  $EA26    Datenblock auf Band schreiben

******************************     Bandpuffer auf Band schreiben

EA15:  20 87 E9  JSR  $E987    Bandpufferadresse laden
EA18:  A9 14     LDA  # $14    Länge des WRITE Vorspanns setzen
EA1A:  85 AB     STA  * $AB    in Z-Page zwischenspeichern

******************************     Datenblock auf Band schreiben

EA1C:  20 E9 E9  JSR  $E9E9    Warte auf "Record & Play" Taste
```

```
EA1F:  B0 7A     BCS  $EA9B    STOP Taste gedrückt, Rücksprung
EA21:  78        SEI           Alle System Interrupts verhindern
EA22:  A9 82     LDA  # $82    IRQ bei Unterlauf des Timers B
EA24:  A2 08     LDX  # $08    Nummer des IRQ Vektors ($EE2E)
EA26:  A0 00     LDY  # $00    Interrupt Maskenregister im CIA
EA28:  8C 1A D0  STY  $D01A    auf #0 setzen (Interrupt disable)
EA2B:  88        DEY           Y-Reg auf $FF herunterzählen  und
EA2C:  8C 19 D0  STY  $D019    Interrupt Request Register setzen
EA2F:  8D 0D DC  STA  $DC0D    IRQ Maske neu setzen
EA32:  AD 0E DC  LDA  $DC0E    CIA Steuerregister A laden, Timer B
EA35:  09 19     ORA  # $19    laden, "One shot" und starten
EA37:  8D 0F DC  STA  $DC0F    Steuerregister B, IRQ an Timer B
EA3A:  29 91     AND  # $91    Zeitvergleichszeiger für Band-
EA3C:  8D 0B 0A  STA  $0A0B    operationen entsprechend setzen
EA3F:  20 EC E7  JSR  $E7EC    Ende der RS-232 Übertragung abwarten
EA42:  AD 11 D0  LDA  $D011    VIC Steuerregister in Akku und in
EA45:  A8        TAY           Y-Reg kopieren
EA46:  29 10     AND  # $10    Bit 4, Bildschirm ein, setzen
EA48:  8D 39 0A  STA  $0A39    Wert im VDC Hilfsspeicher sichern
EA4B:  98        TYA           Alten Wert zurück in den Akku
EA4C:  29 6F     AND  # $6F    Bit 8 des Rastervergleichs löschen
EA4E:  8D 11 D0  STA  $D011    und den Bildschirm abschalten
EA51:  20 74 E5  JSR  $E574    Takt auf 1MHz und Sprites abschalten
EA54:  AD 14 03  LDA  $0314    IRQ-Vektor Lo Adresse in IRQ
EA57:  8D 09 0A  STA  $0A09    Zwischenspeicher für Bandoperationen
EA5A:  AD 15 03  LDA  $0315    IRQ-Vektor Hi Adresse in IRQ
EA5D:  8D 0A 0A  STA  $0A0A    Zwischenspeicher für Bandoperationen
EA60:  20 9B EE  JSR  $EE9B    IRQ-Vektor für Band I/O neu setzen
EA63:  A9 02     LDA  # $02    Anzahl der zu lesenden Datenblöcke
EA65:  85 BE     STA  * $BE    in Z-Page Speicher merken
EA67:  20 5A ED  JSR  $ED5A    Bit Zähler initialisieren, Ser. I/O
EA6A:  A5 01     LDA  * $01    Motor des Kassettenrekorders über
EA6C:  29 1F     AND  # $1F    Einblenden des 4. Bits in das
EA6E:  85 01     STA  * $01    Prozessorport Datenreg. einschalten
EA70:  85 C0     STA  * $C0    Zeiger für Band Motor setzen
EA72:  A2 FF     LDX  # $FF    Zähler für Warteschleife Hi
EA74:  A0 FF     LDY  # $FF    Zähler für Warteschleife Lo
EA76:  88        DEY           Hier werden das X-Reg und Y-Reg
EA77:  D0 FD     BNE  $EA76    von 65535 bis 0 heruntergezählt, um
EA79:  CA        DEX           so die nötige Wartezeit für den
EA7A:  D0 F8     BNE  $EA74    Bandanlauf zu überbrücken
EA7C:  58        CLI           Interrupt für Band I/O freigeben

*******************************  Band I/O Abschluß abwarten

EA7D:  AD 0A 0A  LDA  $0A0A    Band IRQ Vektor mit dem normalen
EA80:  CD 15 03  CMP  $0315    IRQ Zeiger Hi vergleichen
EA83:  18        CLC           Kennzeichen für OK setzen
```

```
EA84:  F0 15     BEQ  $EA9B    IRQ Vektoren gleich, dann fertig
EA86:  20 8F EA  JSR  $EA8F    Prüfe, ob STOP Taste gedrückt
EA89:  20 3D F6  JSR  $F63D    Wenn gedrückt, Flag entsp. setzen
EA8C:  4C 7D EA  JMP  $EA7D    Weiter auf Abschluß warten

******************************  Testen auf STOP-Taste

EA8F:  20 E1 FF  JSR  $FFE1    Kernal STOP: Auf Stop Taste prüfen
EA92:  18        CLC           Kennzeichen für alles OK setzen
EA93:  D0 0B     BNE  $EAA0    STOP nicht gedrückt, dann RTS Exit
EA95:  20 57 EE  JSR  $EE57    Motor aus, normalen IRQ setzen
EA98:  38        SEC           Carry für Fehler-KZ setzen
EA99:  68        PLA           Die Rücksprungadresse vom
EA9A:  68        PLA           Stack zurückholen und löschen
EA9B:  A9 00     LDA  # $00    Code für "Abbruch" in Akku laden
EA9D:  8D 0A 0A  STA  $0A0A    und Kennzeichen f. norm. IRQ setzen
EAA0:  60        RTS           Rücksprung aus dem Unterprogramm

******************************  Kassettensynchronisation vorbereiten

EAA1:  86 B1     STX  * $B1    X-Reg Inhalt in Z-Page sichern
EAA3:  A5 B0     LDA  * $B0    Timing Konstante für Band in Akku
EAA5:  0A        ASL  A        Die Timing Konstante wird mit
EAA6:  0A        ASL  A        dem Faktor 4 multipliziert
EAA7:  18        CLC           Carry für Addition löschen
EAA8:  65 B0     ADC  * $B0    Timing Konstante addieren (entsp.*5)
EAAA:  18        CLC           Carry für Addition löschen
EAAB:  65 B1     ADC  * $B1    Alten X-Reg Inhalt dazuaddieren und
EAAD:  85 B1     STA  * $B1    diesen Wert in Z-Page ablegen
EAAF:  A9 00     LDA  # $00    Low-Value für Timer A laden
EAB1:  24 B0     BIT  * $B0    Prüfe, ob Timing Konstante >128 ist
EAB3:  30 01     BMI  $EAB6    Ja, dann Skip Abgleich
EAB5:  2A        ROL  A        Durch die entsprechende Rotation
EAB6:  06 B1     ASL  * $B1    des Akku Inhalts in Verbindung
EAB8:  2A        ROL  A        mit einer Verschiebung der Band
EAB9:  06 B1     ASL  * $B1    Timing Konstante wird der Init.-Wert
EABB:  2A        ROL  A        für Timer A vervierfacht.
EABC:  AA        TAX           Hi dieses Timer Wertes in X sichern
EABD:  AD 06 DC  LDA  $DC06    Lo Wert des CIA 1 Timers B in Akku
EAC0:  C9 16     CMP  # $16    Veränderung von Timer B Hi bis 63755
EAC2:  90 F9     BCC  $EABD    Ja, dann schleife zur Timer Abfrage
EAC4:  65 B1     ADC  * $B1    Lo für Initialisierung addieren
EAC6:  8D 04 DC  STA  $DC04    und in Timer A Lo setzen
EAC9:  8A        TXA           Hi Wert der Initialisierung in Akku
EACA:  6D 07 DC  ADC  $DC07    mit Überl. zu Timer B Hi addieren
EACD:  8D 05 DC  STA  $DC05    und in Timer A Hi setzen
EAD0:  AD 0B 0A  LDA  $0A0B    Init. Wert aus Band Zeitkonstante
EAD3:  8D 0E DC  STA  $DC0E    zum Starten des Timers A kopieren
```

```
EAD6:  8D 0D 0A   STA  $0A0D     Timer A Flag zurücksetzen
EAD9:  AD 0D DC   LDA  $DC0D     Interrupt Control Register in Akku
EADC:  29 10      AND  # $10     Prüfe, ob negative Flanke an FLAG
EADE:  F0 09      BEQ  $EAE9     Nein, dann warte auf neg. Flanke
EAE0:  A9 EA      LDA  # $EA     Den Inhalt der beiden Z-Page
EAE2:  48         PHA            Speicher $Ea und $E9 als
EAE3:  A9 E9      LDA  # $E9     quasi Rücksprungadresse auf
EAE5:  48         PHA            den System Stapel ablegen
EAE6:  4C C8 EE   JMP  $EEC8     Simulieren des Interrupt-Aufrufs
EAE9:  58         CLI            Alle System Interrupts freigeben
EAEA:  60         RTS            Rücksprung aus dem Unterprogramm

******************************   Interrupt Rout. für Band lesen

EAEB:  AE 07 DC   LDX  $DC07     CIA 1 Timer B Hi in X-Reg
EAEE:  A0 FF      LDY  # $FF     Y-Reg mit High-Value Wert init.
EAF0:  98         TYA            und für Subtraktion in Akku
EAF1:  ED 06 DC   SBC  $DC06     Timer B Lo von #255 subtrahieren
EAF4:  EC 07 DC   CPX  $DC07     Ist Timer B Hi schon vermindert?
EAF7:  D0 F2      BNE  $EAEB     Ja, zurück zum Timervergleich
EAF9:  86 B1      STX  * $B1     Timer B Hi in Z-Page ablegen
EAFB:  AA         TAX            Zeit Lo seit letzter Flanke in X-Reg
EAFC:  8C 06 DC   STY  $DC06     Timer B Lo auf High-Value
EAFF:  8C 07 DC   STY  $DC07     Timer B Hi auf High-Value
EB02:  A9 19      LDA  # $19     Arbeitsmodus für Timer B festlegen
EB04:  8D 0F DC   STA  $DC0F     und Timer B starten
EB07:  AD 0D DC   LDA  $DC0D     Interrupt Control Register in Akku
EB0A:  8D 0C 0A   STA  $0A0C     und in Systemspeicher für Band
EB0D:  98         TYA            Akku mit #255 initialisieren
EB0E:  E5 B1      SBC  * $B1     Timer B Hi von #255 subtrahieren
EB10:  86 B1      STX  * $B1     Verstrichene Zeit in Z-Page sichern
EB12:  4A         LSR  A         Der im Akku gesicherte Wert
EB13:  66 B1      ROR  # $B1     für die vergangene Zeit
EB15:  4A         LSR  A         wird durch den Faktor 4
EB16:  66 B1      ROR  # $B1     dividiert
EB18:  A5 B0      LDA  * $B0     Timing Konstante aus Z-Page holen
EB1A:  18         CLC            Carry für Addition löschen
EB1B:  69 3C      ADC  # $3C     #60 zur Timing-Konstante addieren
EB1D:  C5 B1      CMP  * $B1     Größer als Zeit seit letzter Flanke?
EB1F:  B0 4A      BCS  $EB6B     Ja, dan keine Information, Skip
EB21:  A6 9C      LDX  * $9C     Wurde ein Byte empfangen?
EB23:  F0 03      BEQ  $EB28     Nein, dann Skip
EB25:  4C 1F EC   JMP  $EC1F     Weiter bei Routine Byte empfangen
EB28:  A6 A3      LDX  * $A3     Wurde Byte vollständig gelesen?
EB2A:  30 1B      BMI  $EB47     Ja, dann entspr. auswerten
EB2C:  A2 00      LDX  # $00     Code für kurzen Impuls in X-Reg (0)
EB2E:  69 30      ADC  # $30     Akku für Impulsabfrage setzen
EB30:  65 B0      ADC  * $B0     und Timing Konstante addieren
```

```
EB32:  C5 B1     CMP  * $B1     Kurzer Zeitimpuls empfangen?
EB34:  B0 1C     BCS  $EB52     Ja, dann Skip langen Impuls
EB36:  E8        INX            Code für langen Impuls in X-Reg (1)
EB37:  69 26     ADC  # $26     Akku für Impulsabfrage setzen
EB39:  65 B0     ADC  * $B0     und Timing Konstante addieren
EB3B:  C5 B1     CMP  * $B1     Langen Zeitimpuls empfangen?
EB3D:  B0 17     BCS  $EB56     Ja, dann Skip andere Impulsdauer
EB3F:  69 2C     ADC  # $2C     Prüfe, ob der vergangene Zeit-
EB41:  65 B0     ADC  * $B0     Impuls noch länger war. Wenn ja,
EB43:  C5 B1     CMP  * $B1     so ist es ein Byte-Header Impuls
EB45:  90 03     BCC  $EB4A     Nein, dann Skip Verarbeitung
EB47:  4C CF EB  JMP  $EBCF     Empfangenes Byte verarbeiten
EB4A:  A5 B4     LDA  * $B4     Prüfe, ob der Timer A freigegeben
EB4C:  F0 1D     BEQ  $EB6B     ist. Nein, dann Skip
EB4E:  85 A8     STA  * $A8     Zeiger für "READ ERROR" setzen
EB50:  D0 19     BNE  $EB6B     Sprung zur Timer Interrupt Abfrage
EB52:  E6 A9     INC  * $A9     Zeiger für Impulslängenwechsel +1
EB54:  B0 02     BCS  $EB58     Skip Verminderung des Wechsels
EB56:  C6 A9     DEC  * $A9     Zeiger für Impulslängenwechsel -1
EB58:  38        SEC            Carry für Subtraktion setzen
EB59:  E9 13     SBC  # $13     Vom Abfragewert #19, sowie die
EB5B:  E5 B1     SBC  * $B1     vergangene Zeit subtrahieren
EB5D:  65 92     ADC  * $92     Z-Page Speicher f. Timinig Korrektur
EB5F:  85 92     STA  * $92     Flag addieren und darin abspeichern
EB61:  A5 A4     LDA  * $A4     Das Zero-Page Flag für den Empfang
EB63:  49 01     EOR  # $01     beider Impulse invertieren
EB65:  85 A4     STA  * $A4     und wieder in Z-Page ablegen
EB67:  F0 2B     BEQ  $EB94     Beide Impuls empfangen, dann Skip
EB69:  86 C5     STX  * $C5     Empfangenes Signal in Z-Page sichern
EB6B:  A5 B4     LDA  * $B4     Prüfe, ob Timer A freigegeben ist
EB6D:  F0 22     BEQ  $EB91     Nein, dann Interrupt abschließen
EB6F:  AD 0C 0A  LDA  $0A0C     Hole Inhalt des ICR in Akku
EB72:  29 01     AND  # $01     War es ein Timer A Interrupt?
EB74:  D0 05     BNE  $EB7B     Ja, dann Skip
EB76:  AD 0D 0A  LDA  $0A0D     Prüfe, ob Timer A abgelaufen ist
EB79:  D0 16     BNE  $EB91     Nein, dann Interrupt abschließen
EB7B:  A9 00     LDA  # $00     Das Zero-Page Flag für die
EB7D:  85 A4     STA  * $A4     Impulszählung löschen (Low-Value)
EB7F:  8D 0D 0A  STA  $0A0D     Zeiger für "Timeout" Timer A setzen
EB82:  A5 A3     LDA  * $A3     Prüfe, ob Byte vollständig gelesen
EB84:  10 30     BPL  $EBB6     wurde. Nein, dann Skip
EB86:  30 BF     BMI  $EB47     Ja, dann entspr. verarbeiten
EB88:  A2 A6     LDX  # $A6     Initialisierungswert für Timer A
EB8A:  20 A1 EA  JSR  $EAA1     Band zum Lesen vorbereiten
EB8D:  A5 9B     LDA  * $9B     Z-Page Band Paritätsbyte in Akku
EB8F:  D0 B9     BNE  $EB4A     Nicht Null, dann Paritätsfehler
EB91:  4C 33 FF  JMP  $FF33     Zurück zum Kernal Interrupt
EB94:  A5 92     LDA  * $92     Timing-Korrektur-Zeiger in Akku
```

```
EB96:  F0 07     BEQ  $EB9F      Flag gelöscht, dann Skip
EB98:  30 03     BMI  $EB9D      Kleiner Null, Skip Dec der Konstante
EB9A:  C6 B0     DEC  * $B0      Z-Page Timing Konstante -1
EB9C:  2C        .Byte  $2C      Skip nach $EB9F
EB9D:  E6 B0     INC  * $D0      Z-Page Timing Konstante +1
EB9F:  A9 00     LDA  # $00      Z-Page Zeiger für Timing-Konstanten
EBA1:  85 92     STA  * $92      Korrektur löschen (Low-Value)
EBA3:  E4 C5     CPX  * $C5      Vgl. empf. Impuls mit vorherigem
EBA5:  D0 0F     BNE  $EBB6      Nicht gleich, dann OK und Skip
EBA7:  8A        TXA             Prüfe, ob kurzer Impuls empfangen
EBA8:  D0 A0     BNE  $EB4A      Nein, dann Lesefehler. Skip
EBAA:  A5 A9     LDA  * $A9      Impulslängenwechselzeiger in Akku
EBAC:  30 BD     BMI  $EB6B      Negativer Wert, dann Skip
EBAE:  C9 10     CMP  # $10      Wurden 16 kurze Impulse empfangen?
EBB0:  90 B9     BCC  $EB6B      Nein, dann wie negativer Wert
EBB2:  85 96     STA  * $96      Ja, dann EOB Flag empfangen
EBB4:  B0 B5     BCS  $EB6B      Unbedingter Sprung
EBB6:  8A        TXA             Empfangenes Bit in Akku bringen,
EBB7:  45 9B     EOR  * $9B      mit Band-Parität verknüpfen und
EBB9:  85 9B     STA  * $9B      wieder in Band-Parität speichern
EBBB:  A5 B4     LDA  * $B4      Prüfe, ob Timer A freigegeben ist
EBBD:  F0 D2     BEQ  $EB91      Nein, dann Interrupt beenden
EBBF:  C6 A3     DEC  * $A3      Z-Page Speicher f. Bitzähler -1
EBC1:  30 C5     BMI  $EB88      Paritätsbit empfangen? Ja, dann Skip
EBC3:  46 C5     LSR  * $C5      Nein, dann gelesenes Bit in
EBC5:  66 BF     ROR  # $BF      Z-Page Speicher f. Banddaten gelesen
EBC7:  A2 DA     LDX  # $DA      Initialisierungswert f. Timer A
EBC9:  20 A1 EA  JSR  $EAA1      Kassettensynchronisation vorbereiten
EBCC:  4C 33 FF  JMP  $FF33      Zurück zur IRQ Routine
EBCF:  A5 96     LDA  * $96      Prüfe, ob EOB empfangen wurde
EBD1:  F0 04     BEQ  $EBD7      Nein, Skip Timer Abfrage
EBD3:  A5 B4     LDA  * $B4      Prüfe, ob Timer A freigegeben ist
EBD5:  F0 07     BEQ  $EBDE      Nein, überspringe Bit-Zähler-Test
EBD7:  A5 A3     LDA  * $A3      Prüfe, ob Z-Page Bitzähler negativ
EBD9:  30 03     BMI  $EBDE      Ja, Bytes-Header abwarten
EBDB:  4C 56 EB  JMP  $EB56      Verarb.lang.Impuls, kein Byte-Header
EBDE:  46 B1     LSR  * $B1      Halbiere die vergangene Zeit seit
EBE0:  A9 93     LDA  # $93      der letzten negativen Flanke und
EBE2:  38        SEC             subtrahiere diesen Wert von der
EBE3:  E5 B1     SBC  * $B1      Konstante #147.
EBE5:  65 B0     ADC  * $B0      Z-Page Timing Konstante addieren
EBE7:  0A        ASL  A          und diesen Wert verdoppeln
EBE8:  AA        TAX             Als Init.Wert f. Timer A nach X-Reg
EBE9:  20 A1 EA  JSR  $EAA1      Kassettensynchronisation vorbereiten
EBEC:  E6 9C     INC  * $9C      Setze Z-Page Zeiger:"Byte empfangen"
EBEE:  A5 B4     LDA  * $B4      Prüfe, ob Timer A freigegeben ist
EBF0:  D0 11     BNE  $EC03      Ja, dann Skip
EBF2:  A5 96     LDA  * $96      Prüfe, ob EOB empfangen wurde
```

```
EBF4:  F0 26     BEQ  $EC1C     Nein, zur norm. IRQ-Routine
EBF6:  85 A8     STA  * $A8     Z-Page Zeiger f. Lesefehler setzen
EBF8:  A9 00     LDA  # $00     Z-Page Speicher für EOB-Kennzeichen
EBFA:  85 96     STA  * $96     löschen (Low-Value)
EBFC:  A9 81     LDA  # $81     Codewert f.Timer A Interruptfreigabe
EBFE:  8D 0D DC  STA  $DC0D     Interrupt f. Timer A freigeben
EC01:  85 B4     STA  * $B4     Setze Z-Page Flag f. Timer A möglich
EC03:  A5 96     LDA  * $96     Das Z-Page Flag für empfangenes EOB
EC05:  85 B5     STA  * $B5     in Flag f. gültiges EOB kopieren
EC07:  F0 09     BEQ  $EC12     Kein EOB Kennzeichen, dann Skip
EC09:  A9 00     LDA  # $00     Steuercode für Timer A disable
EC0B:  85 B4     STA  * $B4     in entspr. Z-Page Zeiger bringen
EC0D:  A9 01     LDA  # $01     Steuercode zum Sperren des Timers A
EC0F:  8D 0D DC  STA  $DC0D     Interrupts ins CIA Steuerregister
EC12:  A5 BF     LDA  * $BF     Z-Page Shift-Register für READ in
EC14:  85 BD     STA  * $BD     Z-Page Speicher f. gelesenes Byte
EC16:  A5 A8     LDA  * $A8     Z-Page Zeiger für Lesefehler mit
EC18:  05 A9     ORA  * $A9     Impulslängenwechselzeiger verknüpfen
EC1A:  85 B6     STA  * $B6     in Fehlercode des Bytes ablegen
EC1C:  4C 33 FF  JMP  $FF33     Zurück zum normalen IRQ-Aufruf
EC1F:  20 5A ED  JSR  $ED5A     Bitzähler für serielle Ausgabe setzen
EC22:  85 9C     STA  * $9C     Zeiger: "Byte empfangen" zurücksetzen
EC24:  A2 DA     LDX  # $DA     Initialisierungswert für Timer A
EC26:  20 A1 EA  JSR  $EAA1     Kassettensynchronisation vorbereiten
EC29:  A5 BE     LDA  * $BE     Prüfe, ob Anzahl der verbliebenen
EC2B:  F0 02     BEQ  $EC2F     Blöcke Null ist. Ja, dann Skip
EC2D:  85 A7     STA  * $A7     Blockzahl z. Lesen neu setzen
EC2F:  A9 0F     LDA  # $0F     Maskenwert für Zählung vor dem Lesen
EC31:  24 AA     BIT  * $AA     Prüfe Zeiger für Lesen von Band
EC33:  10 17     BPL  $EC4C     Wenn alle Zeichen empfangen, Ende
EC35:  A5 B5     LDA  * $B5     Prüfe, ob gültiges EOB empfangen
EC37:  D0 0C     BNE  $EC45     Ja, dann Skip
EC39:  A6 BE     LDX  * $BE     Ist die Anzahl der zum Lesen
EC3B:  CA        DEX            verbliebenen Blöcke = 1?
EC3C:  D0 0B     BNE  $EC49     Nein, zum normalen IRQ-Aufruf
EC3E:  A9 08     LDA  # $08     Bit 3 für "Long Block" in A setzen
EC40:  20 57 F7  JSR  $F757     System Statuszeiger neu setzen
EC43:  D0 04     BNE  $EC49     Unbed. Sprung zur normalen IRQ-Rout.
EC45:  A9 00     LDA  # $00     Z-Page Zeiger für "Lesen von Band"
EC47:  85 AA     STA  * $AA     auf "Abtastung" setzen (Low-Value)
EC49:  4C 33 FF  JMP  $FF33     Zurück zur normalen IRQ-Routine
EC4C:  70 31     BVS  $EC7F     Skip bei Bandlesezeiger auf "Lesen"
EC4E:  D0 18     BNE  $EC68     Skip bei Bandlesezeiger auf "Zählen"
EC50:  A5 B5     LDA  * $B5     Prüfe, ob EOB empfangen wurde
EC52:  D0 F5     BNE  $EC49     Ja, zurück zur normalen IRQ-Routine
EC54:  A5 B6     LDA  * $B6     Prüfe,ob Byte-Lesefehler aufgetreten
EC56:  D0 F1     BNE  $EC49     Ja, zurück zur normalen IRQ-Routine
EC58:  A5 A7     LDA  * $A7     Hole Zahl d. noch zu lesenden Blocks
```

```
EC5A:  4A        LSR  A         und schiebe Bit 0 ins Carry Flag
EC5B:  A5 BD     LDA  * $BD     Hole gelesenes Byte aus Z-Page
EC5D:  30 03     BMI  $EC62     Ist es ein Zählbyte, dann Skip
EC5F:  90 18     BCC  $EC79     Noch mehr als ein Block lesen, Skip
EC61:  18        CLC            Carry Flag Zeiger zurücksetzen
EC62:  B0 15     BCS  $EC79     Skip, wenn nur ein Block zu lesen
EC64:  29 0F     AND  # $0F     Obere Tetrade (Bit 4-7) ausmaskieren
EC66:  85 AA     STA  * $AA     Als Zählwert abspeichern, Zähler -1
EC68:  C6 AA     DEC  * $AA     und prüfe, ob alle Sync.-Bytes
EC6A:  D0 DD     BNE  $EC49     empfangen. Nein, zum normalen IRQ
EC6C:  A9 40     LDA  # $40     Setze Bit 6 im Akku und den Z-Page
EC6E:  85 AA     STA  * $AA     Bandlesezeiger auf KZ: "Lesen"
EC70:  20 51 ED  JSR  $ED51     Ein-/Ausgabe Startadresse kopieren
EC73:  A9 00     LDA  # $00     Z-Page Zeiger für Leseprüfsumme
EC75:  85 AB     STA  * $AB     löschen (auf Low-Value setzen)
EC77:  F0 D0     BEQ  $EC49     Zurück zur normalen IRQ-Routine
EC79:  A9 80     LDA  # $80     Setze Bit 7 im Akku  und den Z-Page
EC7B:  85 AA     STA  * $AA     Bandlesezeiger auf KZ: "Ende"
EC7D:  D0 CA     BNE  $EC49     Zurück zur normalen !RQ-Routine
EC7F:  A5 B5     LDA  * $B5     Prüfe, ob EOB Kennzeichen gesetzt
EC81:  F0 0A     BEQ  $EC8D     Nein, dann Skip
EC83:  A9 04     LDA  # $04     Bit 2 für kurzen Block in A setzen
EC85:  20 57 F7  JSR  $F757     System Statuszeiger neu setzen
EC88:  A9 00     LDA  # $00     Code für Lesezeiger auf "Abtasten"
EC8A:  4C 0C ED  JMP  $ED0C     setzen und unbedingt springen
EC8D:  20 B7 EE  JSR  $EEB7     Prüfe, ob Ende schon erreicht
EC90:  90 03     BCC  $EC95     Nein, dann normal weiter
EC92:  4C 0A ED  JMP  $ED0A     Zum Read Ende für einen Block
EC95:  A6 A7     LDX  * $A7     Ist die Anzahl der zum Lesen
EC97:  CA        DEX            verbleibenden Blöcke = 1?
EC98:  F0 2E     BEQ  $ECC8     Ja, dann Pass 2 (Korrekturpass)
EC9A:  A5 93     LDA  * $93     Prüfe, ob Verify KZ gesetzt ist
EC9C:  F0 0D     BEQ  $ECAB     Nein, dann Skip
EC9E:  A0 00     LDY  # $00     Displ. für Vgl. auf #0 setzen
ECA0:  20 CC F7  JSR  $F7CC     FETCH Routine für LSV Aufrufe
ECA3:  C5 BD     CMP  * $BD     mit gelesenem Byte vergleichen
ECA5:  F0 04     BEQ  $ECAB     Beide gleich, dann OK und Skip
ECA7:  A9 01     LDA  # $01     Code für gefundenen Zeichen-Lese-
ECA9:  85 B6     STA  * $B6     fehler in Z-Page Band-Hilfszeiger
ECAB:  A5 B6     LDA  * $B6     Teste Band-Hilfszeiger für Fehler
ECAD:  F0 4C     BEQ  $ECFB     Kein Fehler aufgetreten, dann Skip
ECAF:  A2 3D     LDX  # $3D     Prüfe, ob beim Lesen schon 31
ECB1:  E4 9E     CPX  * $9E     Lesefehler aufgetreten sind
ECB3:  90 3F     BCC  $ECF4     Ja, dann nicht korrigierbar, Skip
ECB5:  A6 9E     LDX  * $9E     Displ. für Adr. Lesefehler im Stack
ECB7:  A5 AD     LDA  * $AD     Hole Adreßbyte des Fehlers Lo
ECB9:  9D 01 01  STA  $0101,X   und sichere Fehleradresse im Stack
ECBC:  A5 AC     LDA  * $AC     Hole Adreßbyte des Fehlers Hi
```

```
ECBE:  9D 00 01   STA  $0100,X     und sichere Fehleradresse im Stack
ECC1:  E8         INX              Fehleradresse-Displacementzeiger +
ECC2:  E8         INX              Fehlerzahl-Zähler um 2 erhöhen
ECC3:  86 9E      STX  * $9E       und wieder im Fehlerzähler ablegen
ECC5:  4C FB EC   JMP  $ECFB       Weiter, wie kein Fehler aufgetreten
ECC8:  A6 9F      LDX  * $9F       Prüfe, ob bereits alle Lesefehler
ECCA:  E4 9E      CPX  * $9E       korrigiert sind
ECCC:  F0 37      BEQ  $ED05       Ja, dann weiter
ECCE:  A5 AC      LDA  * $AC       Hole aktuelles Adreßbyte Lo-Wert
ECD0:  DD 00 01   CMP  $0100,X     Mit Fehleradreßbyte Lo vergleichen
ECD3:  D0 30      BNE  $ED05       Nicht gleich, dann Skip
ECD5:  A5 AD      LDA  * $AD       Hole aktuelles Adreßbyte Hi-Wert
ECD7:  DD 01 01   CMP  $0101,X     Mit Fehleradreßbyte Hi vergleichen
ECDA:  D0 29      BNE  $ED05       Nicht gleich, dann Skip
ECDC:  E6 9F      INC  * $9F       Den Z-Page Korrekturzähler für den
ECDE:  E6 9F      INC  * $9F       Pass 2 um 2 erhöhen
ECE0:  A5 93      LDA  * $93       Prüfe ob Verify KZ gesetzt ist
ECE2:  F0 0C      BEQ  $ECF0       Nein, dann Skip
ECE4:  A0 00      LDY  # $00       Displacement für FETCH Routine
ECE6:  20 CC F7   JSR  $F7CC       FETCH Routine für LSV Aufrufe
ECE9:  C5 BD      CMP  * $BD       Gelesenes Byte gleich Speicherbyte?
ECEB:  F0 18      BEQ  $ED05       Ja, dann Skip
ECED:  C8         INY              Displacementzeiger um 1 erhöhen
ECEE:  84 B6      STY  * $B6       und in Z-Page Fehlerzeiger bringen
ECF0:  A5 B6      LDA  * $B6       Prüfe, ob Fehler aufgetreten
ECF2:  F0 07      BEQ  $ECFB       Nein, dann Skip
ECF4:  A9 10      LDA  # $10       Bit 4 setzen (nicht kor. Lesefehler)
ECF6:  20 57 F7   JSR  $F757       System Statuszeiger neu setzen
ECF9:  D0 0A      BNE  $ED05       Unbedingter Sprung
ECFB:  A5 93      LDA  * $93       Prüfe, ob Verify KZ gesetzt ist
ECFD:  D0 06      BNE  $ED05       Ja, dann Skip
ECFF:  A8         TAY              Displacementzeiger auf #0 setzen
ED00:  A5 BD      LDA  * $BD       Gelesenes Byte in Akku holen
ED02:  20 BC F7   JSR  $F7BC       STASH Routine für LSV Routinen
ED05:  20 C1 EE   JSR  $EEC1       Ein-/Ausgabe Startadresse erhöhen
ED08:  D0 44      BNE  $ED4E       Zurück zur normalen IRQ-Routine
ED0A:  A9 80      LDA  # $80       Code für Lesezeiger auf "Ende"
ED0C:  85 AA      STA  * $AA       Bandlesezeiger entsp. Akku setzen
ED0E:  78         SEI              Alle System Interrupts verhindern
ED0F:  A2 01      LDX  # $01       Codewert für Interrupt des Timers A
ED11:  8E 0D DC   STX  $DC0D       verhindern ins ICR Register
ED14:  AE 0D DC   LDX  $DC0D       Interrupt Zeiger zurücksetzen
ED17:  A6 BE      LDX  * $BE       Prüfe, ob die Anzahl der noch zu
ED19:  CA         DEX              verarbeitenden Blöcke Null ist.
ED1A:  30 02      BMI  $ED1E       Ja, dann Skip
ED1C:  86 BE      STX  * $BE       Neue Anzahl in Z-Page sichern
ED1E:  C6 A7      DEC  * $A7       Z-Page Blockzähler um 1 vermindern
ED20:  F0 08      BEQ  $ED2A       Blockzähler = 0, dann Skip
```

```
ED22:  A5 9E     LDA  * $9E     Prüfe, ob Fehler in Pass 1
ED24:  D0 28     BNE  $ED4E     augetreten ist. Ja, dann Skip
ED26:  85 BE     STA  * $BE     Anzahl der zu verarbeitenden Bl.: #0
ED28:  F0 24     BEQ  $ED4E     Zurück zur normalen IRQ-Routine
ED2A:  20 57 EE  JSR  $EE57     Routine: Ende Band I/O
ED2D:  20 51 ED  JSR  $ED51     Startadresse in Ladezeiger kopieren
ED30:  A0 00     LDY  # $00     Den Zero-Page Zeiger für Prüfsumme
ED32:  84 AB     STY  * $AB     löschen. Displ. auf Null setzen
ED34:  20 CC F7  JSR  $F7CC     FETCH Routine für LSV Operationen
ED37:  45 AB     EOR  * $AB     Speicherbyte m. Prüfsumme verknüpfen
ED39:  85 AB     STA  * $AB     und im Prüfsummenzeiger ablegen
ED3B:  20 C1 EE  JSR  $EEC1     Ein-/Ausgabe Startadresse erhöhen
ED3E:  20 B7 EE  JSR  $EEB7     Prüft auf Erreichen der Endadresse
ED41:  90 F1     BCC  $ED34     Nicht Endadresse, dann weiter
ED43:  A5 AB     LDA  * $AB     Vergleiche die gebildete Prüfsumme
ED45:  45 BD     EOR  * $BD     mit der gelesenen Prüfsumme
ED47:  F0 05     BEQ  $ED4E     Gleich, dann OK und weiter
ED49:  A9 20     LDA  # $20     Bit 5 setzen (Prüfsummenfehler)
ED4B:  20 57 F7  JSR  $F757     System Statuszeiger neu setzen
ED4E:  4C 33 FF  JMP  $FF33     Zurück zur normalen IRQ-Routine

******************************  Ein-/Ausgabe Startadresse kopieren

ED51:  A5 C2     LDA  * $C2     Hole Ein-/Ausgabe Startadresse Hi
ED53:  85 AD     STA  * $AD     Sichere Hi Wert in Z-Page $AD
ED55:  A5 C1     LDA  * $C1     Hole Ein-/Ausgabe Startadresse Lo
ED57:  85 AC     STA  * $AC     Sichere Lo Wert in Z-Page $AC
ED59:  60        RTS            Rücksprung aus dem Unterprogramm

******************************  Setze Bitzähler für serielle Ausgabe

ED5A:  A9 08     LDA  # $08     Zähler für 8 zu übertragende Bits
ED5C:  85 A3     STA  * $A3     in Z-Page initialisieren
ED5E:  A9 00     LDA  # $00     Das Hi-Byte des 2 Byte Z-Page
ED60:  85 A4     STA  * $A4     Zählers auf $00 setzen
ED62:  85 A8     STA  * $A8     Band Lesefehler Flag löschen
ED64:  85 9B     STA  * $9B     Paritätsbyte v. Band initialisieren
ED66:  85 A9     STA  * $A9     Band 0 Leseflag initialisieren
ED68:  60        RTS            Rücksprung aus dem Unterprogramm

******************************  Ein Bit auf Band schreiben

ED69:  A5 BD     LDA  * $BD     Auszugebendes Bit aus Z-Page in Akku
ED6B:  4A        LSR  A         und auszugebendes Bit (0) ins Carry
ED6C:  A9 60     LDA  # $60     Zeit für "0-Bit" setzen
ED6E:  90 02     BCC  $ED72     Timer setzen und ausgeben
ED70:  A9 B0     LDA  # $B0     Zeit für "1-Bit" setzen
ED72:  A2 00     LDX  # $00     Low-Value für Timer Hi-Bytes
```

```
ED74:  8D 06 DC   STA  $DC06      CIA 1 Timer B Low Byte auf Bitzeit
ED77:  8E 07 DC   STX  $DC07      CIA 1 Timer B Hi Byte auf Low Value
ED7A:  AD 0D DC   LDA  $DC0D      Interrupt Flag löschen
ED7D:  A9 19      LDA  # $19      Timer B laden, "One shot" und CIA
ED7F:  8D 0F DC   STA  $DC0F      Steuerregister starten. IRQ an Timer
ED82:  A5 01      LDA  * $01      Invertierungswert für Ausgabebit
ED84:  49 08      EOR  # $08      im Prozessorport invertieren und
ED86:  85 01      STA  * $01      wieder in Prozessorport bringen
ED88:  29 08      AND  # $08      augenblicklichen Pegel merken
ED8A:  60         RTS             Rücksprung aus dem Unterprogramm

******************************    Zeiger f. "Block geschrieben" setzen

ED8B:  38         SEC             Carry für Rotation setzen
ED8C:  66 B6      ROR  * $B6      "Block geschrieben" Flag negieren
ED8E:  30 3C      BMI  $EDCC      Interrupt Rücksprung

******************************    Interrupt Routine für Band schreiben

ED90:  A5 A8      LDA  * $A8      Prüfe, ob Byte Impuls geschrieben
ED92:  D0 12      BNE  $EDA6      Ja, dann Skip Byte Impuls schreiben
ED94:  A9 10      LDA  # $10      Lo-Wert für Byte-Frequenz in Akku
ED96:  A2 01      LDX  # $01      Hi-Wert für Byte-Frequenz in X-Reg
ED98:  20 74 ED   JSR  $ED74      "Byte" Impuls auf Band schreiben
ED9B:  D0 2F      BNE  $EDCC      Wenn erste Halbwelle, zum norm. IRQ
ED9D:  E6 A8      INC  * $A8      Zeiger für Impuls geschrieben setzen
ED9F:  A5 B6      LDA  * $B6      Prüfe "Block geschrieben" Zeiger
EDA1:  10 29      BPL  $EDCC      Ja, dann zurück zum normalen IRQ
EDA3:  4C 1B EE   JMP  $EE1B      Block fertig, dann Write fortsetzen
EDA6:  A5 A9      LDA  * $A9      Prüfe, ob langer Impuls geschrieben
EDA8:  D0 09      BNE  $EDB3      Ja, dann Skip langen Impuls
EDAA:  20 70 ED   JSR  $ED70      Langen Impuls auf Band schreiben
EDAD:  D0 1D      BNE  $EDCC      Wenn erste Halbwelle, zum norm. IRQ
EDAF:  E6 A9      INC  * $A9      Zeiger für Impuls geschrieben setzen
EDB1:  D0 19      BNE  $EDCC      Zurück zur normalen IRQ-Routine
EDB3:  20 69 ED   JSR  $ED69      Ein Bit auf Band schreiben
EDB6:  D0 14      BNE  $EDCC      Wenn erste Halbwelle, zum norm. IRQ
EDB8:  A5 A4      LDA  * $A4      Den Z-Page Bit-Impuls
EDBA:  49 01      EOR  # $01      Zeiger invertieren und
EDBC:  85 A4      STA  * $A4      wieder zurückspeichern
EDBE:  F0 0F      BEQ  $EDCF      Wenn #0, beide Impulse geschrieben
EDC0:  A5 BD      LDA  * $BD      Das Bit 0 des Zero-Page Bit-Shift
EDC2:  49 01      EOR  # $01      Speichers invertieren
EDC4:  85 BD      STA  * $BD      und wieder zurückspeichern
EDC6:  29 01      AND  # $01      Aktuelles Bit eliminieren und mit
EDC8:  45 9B      EOR  * $9B      Paritäts-Bit des Bytes verknüpfen
EDCA:  85 9B      STA  * $9B      und in Paritäts Flag zurückspeichern
EDCC:  4C 33 FF   JMP  $FF33      Zurück zur normalen IRQ-Routine
```

```
EDCF:  46 BD     LSR  * $BD    Bit herausschieben und den
EDD1:  C6 A3     DEC  * $A3    Z-Page Bitzähler um 1 vermindern
EDD3:  A5 A3     LDA  * $A3    Ist Ende schon erreicht?
EDD5:  F0 3B     BEQ  $EE12    Ja, dann bilde Prüfsumme. Skip
EDD7:  10 F3     BPL  $EDCC    Nein, dann zurück zum norm. IRQ
EDD9:  20 5A ED  JSR  $ED5A    Setze Bitzähler für serielle Ausgabe
EDDC:  58        CLI           Alle System Interrupts freigeben
EDDD:  A5 A5     LDA  * $A5    Prüfe, ob Symc. Bytes geschrieben
EDDF:  F0 12     BEQ  $EDF3    Ja, dann Skip
EDE1:  A2 00     LDX  # $00    Den Prüfsummenspeicher für den
EDE3:  86 C5     STX  * $C5    Lesepuffer löschen (Low-Value)
EDE5:  C6 A5     DEC  * $A5    Sync. Zähler um 1 vermindern
EDE7:  A6 BE     LDX  * $BE    Prüfe, ob der erste Block
EDE9:  E0 02     CPX  # $02    schon geschrieben ist
EDEB:  D0 02     BNE  $EDEF    Nein, dann Skip
EDED:  09 80     ORA  # $80    Bit 7 im Sync. Byte einblenden
EDEF:  85 BD     STA  * $BD    und in Z-Page Bit-Shift Speicher
EDF1:  D0 D9     BNE  $EDCC    Zurück zur normalen IRQ-Routine
EDF3:  20 B7 EE  JSR  $EEB7    Prüfe auf Erreichen der Endadresse
EDF6:  90 0A     BCC  $EE02    Nicht erreicht, weiter schreiben
EDF8:  D0 91     BNE  $ED8B    "Block geschrieben" Zeiger setzen
EDFA:  E6 AD     INC  * $AD    Aktuelles Adreßbyte +1
EDFC:  A5 C5     LDA  * $C5    Hole Puffer Prüfsumme aus Zero-Page
EDFE:  85 BD     STA  * $BD    Sichere Wert im Bit-Shift Speicher
EE00:  B0 CA     BCS  $EDCC    Zurück zur normalen IRQ-Routine
EE02:  A0 00     LDY  # $00    Displacementzeiger auf #0 setzen
EE04:  20 CC F7  JSR  $F7CC    FETCH Routine für LSV Operationen
EE07:  85 BD     STA  * $BD    Bringe Zeichen in Bit-Shift Speicher
EE09:  45 C5     EOR  * $C5    Mit Prüfsummenspeicher verknüpfen
EE0B:  85 C5     STA  * $C5    und wieder zurückspeichern
EE0D:  20 C1 EE  JSR  $EEC1    Ein-/Ausgabe Startadresse erhöhen
EE10:  D0 BA     BNE  $EDCC    Zurück zur normalen IRQ-Routine
EE12:  A5 9B     LDA  * $9B    Paritäts-Bit des Bytes aus Z-Page
EE14:  49 01     EOR  # $01    invertieren und in den Bit-Shift
EE16:  85 BD     STA  * $BD    Speicher kopieren
EE18:  4C 33 FF  JMP  $FF33    Zurück zur normalen IRQ-Routine
EE1B:  C6 BE     DEC  * $BE    Prüfe, ob alle Blöcke geschrieben
EE1D:  D0 03     BNE  $EE22    Nein, dann Skip
EE1F:  20 B0 EE  JSR  $EEB0    Rekordermotor ausschalten
EE22:  A9 50     LDA  # $50    Z-Page Zähler für die Länge
EE24:  85 A7     STA  * $A7    der "Shorts" neu initialisieren
EE26:  A2 08     LDX  # $08    Displacement für IRQ #1 (Write)
EE28:  78        SEI           Alle System Interrupts verhindern
EE29:  20 9B EE  JSR  $EE9B    Setzen der IRQ-Vektoren
EE2C:  D0 EA     BNE  $EE18    Zurück zur normalen IRQ-Routine
```

```
*****************************    Schreiben des Headers (IRQ #1)

EE2E:  A9 78      LDA  # $78     Code für "Header Impuls" in Akku
EE30:  20 72 ED   JSR  $ED72     und Header Impuls schreiben
EE33:  D0 E3      BNE  $EE18     Wenn erste Halbwelle, zum norm. IRQ
EE35:  C6 A7      DEC  * $A7     Headerzähler um 1 vermindern
EE37:  D0 DF      BNE  $EE18     Kein Ende,dann zur norm. IRQ-Routine
EE39:  20 5A ED   JSR  $ED5A     Bitzähler f. serielle Ausgabe setzen
EE3C:  C6 AB      DEC  * $AB     Dauer der Shorts vor und nach Daten
EE3E:  10 D8      BPL  $EE18     Kein Ende,dann zur norm. IRQ-Routine
EE40:  A2 0A      LDX  # $0A     Displacement für IRQ #2 (Write)
EE42:  20 9B EE   JSR  $EE9B     Setzen des IRQ Vektors
EE45:  58         CLI            Alle System Interrupts freigeben
EE46:  E6 AB      INC  * $AB     Dauer der Shorts um 1 vermindern
EE48:  A5 BE      LDA  * $BE     Prüfe, ob alle Blöcke geschrieben
EE4A:  F0 49      BEQ  $EE95     Ja, dann Skip
EE4C:  20 51 ED   JSR  $ED51     Ein-/Ausgabe Endadresse kopieren
EE4F:  A2 09      LDX  # $09     Den Z-Page Zeiger für die Sync.
EE51:  86 A5      STX  * $A5     mit #9 neu setzen und den "Block
EE53:  86 B6      STX  * $B6     geschrieben" Zeiger zurücksetzen
EE55:  D0 82      BNE  $EDD9     Unbedingter Sprung

*****************************    Rekorderbetrieb beenden

EE57:  08         PHP            Prozessor Status auf Stack retten
EE58:  78         SEI            Alle System Interrupts verhindern
EE59:  AD 11 D0   LDA  $D011     Inhalt des VIC Steuerregisters in A
EE5C:  0D 39 0A   ORA  $0A39     Mit VDC Hilfszeiger verknüpfen
EE5F:  29 7F      AND  # $7F     Bildschirm wieder einschalten
EE61:  8D 11 D0   STA  $D011     und Wert in VIC-Reg zurückschreiben
EE64:  2C 3A 0A   BIT  $0A3A     Prüfe IRQ Sicherungsspeicher
EE67:  30 16      BMI  $EE7F     Bit 7 gesetzt, dann Skip
EE69:  2C 37 0A   BIT  $0A37     Prüfe Taktfrequenz Sicherungsspeich.
EE6C:  10 11      BPL  $EE7F     Bit 7 gelöscht, dann kein Update
EE6E:  AD 38 0A   LDA  $0A38     Hole gesicherten Status für Sprites
EE71:  8D 15 D0   STA  $D015     und setze Sprite-Anzeige Register
EE74:  AD 37 0A   LDA  $0A37     Hole gesicherte Taktfrequenz und
EE77:  8D 30 D0   STA  $D030     setzte System auf alten Wert zurück
EE7A:  A9 00      LDA  # $00     Zwischenspeicher für Sicherung der
EE7C:  8D 37 0A   STA  $0A37     System Taktfrequenz wieder löschen
EE7F:  20 B0 EE   JSR  $EEB0     Rekordermotor ausschalten
EE82:  20 B8 E1   JSR  $E1B8     Timing und CIAs auf Standard setzen
EE85:  AD 0A 0A   LDA  $0A0A     Ist Interruptvektor auf Standard?
EE88:  F0 09      BEQ  $EE93     Ja, dann Exit
EE8A:  8D 15 03   STA  $0315     System IRQ-Vektor Hi auf Standard
EE8D:  AD 09 0A   LDA  $0A09     gesicherte IRQ Lo Adresse holen
EE90:  8D 14 03   STA  $0314     System IRQ-Vektor Lo auf Standard
EE93:  28         PLP            Prozessor Status wieder zurückholen
```

```
EE94:  60          RTS            Rücksprung aus dem Unterprogramm

******************************    Bandbetrieb abschließen

EE95:  20 57 EE    JSR  $EE57     Rekorderbetrieb beenden
EE98:  4C 33 FF    JMP  $FF33     Zurück zur normalen IRQ Routine

******************************    Setzen des IRQ-Vektors

EE9B:  BD A0 EE    LDA  $EEA0,X   X-indizierte IRQ Lo-Adresse aus Tab.
EE9E:  8D 14 03    STA  $0314     in System IRQ Vektor Lo kopieren
EEA1:  BD A1 EE    LDA  $EEA1,X   X-indizierte IRQ Hi-Adresse aus Tab.
EEA4:  8D 15 03    STA  $0315     in System IRQ Vektor Hi kopieren
EEA7:  60          RTS            Rücksprung aus dem Unterprogramm

******************************    Tabelle der IRQ-Vektoren

EEA8:  2E EE       ($EE2E)        IRQ #1: Write auf Band (Header)
EEAA:  90 ED       ($ED90)        IRQ #2: Write auf Band (Puffer)
EEAC:  65 FA       ($FA65)        Normal IRQ für Tastaturabfrage
EEAE:  EB EA       ($EAEB)        IRQ für Lesen vom Band

******************************    Rekordermotor ausschalten

EEB0:  A5 01       LDA  * $01     Status des Prozessorport Datenreg.
EEB2:  09 20       ORA  # $20     in Akku, Bit 5 einblenden und somit
EEB4:  85 01       STA  * $01     den Rekordermotor ausschalten
EEB6:  60          RTS            Rücksprung aus dem Unterprogramm

******************************    Prüft auf Erreichen der Endadresse
                                  Wenn Endadresse > Startadresse C=0

EEB7:  38          SEC            Carry für Subtraktion setzen
EEB8:  A5 AC       LDA  * $AC     Lo der E/A Startadresse in Akku
EEBA:  E5 AE       SBC  * $AE     Davon Lo der E/A Endadresse subtr.
EEBC:  A5 AD       LDA  * $AD     Hi der E/A Startadresse in Akku
EEBE:  E5 AF       SBC  * $AF     Davon Hi der E/A Endadresse subtr.
EEC0:  60          RTS            Rücksprung aus dem Unterprogramm

******************************    Ein-/Ausgabe Startadresse erhöhen

EEC1:  E6 AC       INC  * $AC     Lo Wert der E-/A Startadresse + 1
EEC3:  D0 02       BNE  $EEC7     Kein Überlauf im Lo Wert, dann Exit
EEC5:  E6 AD       INC  * $AD     Hi Wert der E-/A Startadresse + 1
EEC7:  60          RTS            Rücksprung aus dem Unterprogramm
```

```
*****************************    Lösche Break Flag in Prozessor Stat.

EEC8:  08        PHP             Prozessor Status auf Stack bringen
EEC9:  68        PLA             und zurück in den Akku kopieren
EECA:  29 EF     AND  # $EF      Break-Flag darin löschen
EECC:  48        PHA             und Status zurück auf Stack
EECD:  4C 17 FF  JMP  $FF17      Sprung zur Kernal IRQ Routine

*****************************    Prüfe Kassettenrekorder Tasten (IRQ)

EED0:  A5 01     LDA  * $01      Hole Prozessorport Datenregister
EED2:  29 10     AND  # $10      und prüfe, ob Taste gedrückt
EED4:  F0 0A     BEQ  $EEE0      Keine Taste gedrückt, dann Exit
EED6:  A0 00     LDY  # $00      Kennzeichen für Kassettenrekorder
EED8:  84 C0     STY  * $C0      OFF in Z-Page Bandflag neu setzen
EEDA:  A5 01     LDA  * $01      Hole Prozessorport Datenregister
EEDC:  09 20     ORA  # $20      und setze Bit für Motor ausschalten
EEDE:  D0 08     BNE  $EEE8      Unbedingter Sprung
EEE0:  A5 C0     LDA  * $C0      Prüfe Z-Page Bandflag für Motor
EEE2:  D0 06     BNE  $EEEA      Wenn Motor eingeschaltet, dann Skip
EEE4:  A5 01     LDA  * $01      Hole Prozessorport Datenregister
EEE6:  29 DF     AND  # $DF      und lösche Bit für Motor einschalten
EEE8:  85 01     STA  * $01      In Prozessorport zurückschreiben
EEEA:  60        RTS             Rücksprung aus dem Unterprogramm

*****************************    Kernal Routine: GETIN
                                 Ein Zeichen einlesen

EEEB:  A5 99     LDA  * $99      Lade Akku mit aktuellem Eingabegerät
EEED:  D0 0A     BNE  $EEF9      Nicht Tastatur, dann weiter
EEEF:  A5 D0     LDA  * $D0      Anzahl der Zeichen im Tastaturpuffer
EEF1:  05 D1     ORA  * $D1      mit Funktionstastenzeiger verknüpfen
EEF3:  F0 0F     BEQ  $EF04      Kein Zeichen da, dann "OK" Exit
EEF5:  78        SEI             Alle System Interrupts verhindern
EEF6:  4C 06 C0  JMP  $C006      Zeichen aus Tastaturpuffer holen

*****************************    GETIN Auswertung nicht von Tastatur

EEF9:  C9 02     CMP  # $02      Prüfe, ob RS-232 d. Eingabegerät ist
EEFB:  D0 18     BNE  $EF15      Nicht RS-232, dann zur BASIN Routine
EEFD:  84 97     STY  * $97      Aktuellen Inhalt des Y-Reg sichern
EEFF:  20 CE E7  JSR  $E7CE      GETIN Routine von RS-232
EF02:  A4 97     LDY  * $97      Alten Inhalt des Y-Reg zurückholen
EF04:  18        CLC             Kennzeichen für alles "OK" setzen
EF05:  60        RTS             Rücksprung aus dem Unterprogramm
```

```
******************************   Kernal Routine: BASIN
                                 Zeichen lesen

EF06:  A5 99     LDA  * $99      Lade Akku mit aktuellem Eingabegerät
EF08:  D0 0B     BNE  $EF15      Nicht Tastatur, dann weiter
EF0A:  A5 EC     LDA  * $EC      Hole aktuelle Cursor Spalte in Akku
EF0C:  85 E9     STA  * $E9      In Z-Page Start der Eingabespalte
EF0E:  A5 EB     LDA  * $EB      Hole aktuelle Cursor Zeile in Akku
EF10:  85 E8     STA  * $E8      In Z-Page Start der Eingabezeile
EF12:  4C 09 C0  JMP  $C009      Zeichen vom Bildschirm holen
EF15:  C9 03     CMP  # $03      Prüfe,ob Eingabegerät Bildschirm ist
EF17:  D0 09     BNE  $EF22      Nicht Bildschirm, dann weiter
EF19:  85 D6     STA  * $D6      In Z-Page Zeiger für Input/Get
EF1B:  A5 E7     LDA  * $E7      Rechte Fenstergrenze in Akku laden
EF1D:  85 EA     STA  * $EA      In Z-Page f. Ende der Eingabezeile
EF1F:  4C 09 C0  JMP  $C009      Zeichen vom Bildschirm holen
EF22:  B0 38     BCS  $EF5C      Gerät > 3, lese Zeichen vom IEC Bus
EF24:  C9 02     CMP  # $02      Eingabegerät 2 (RS-232) gesetzt?
EF26:  F0 3F     BEQ  $EF67      Ja, dann Zeichen von RS-232 holen
EF28:  86 97     STX  * $97      Aktuellen Inhalt des X-Reg retten
EF2A:  20 48 EF  JSR  $EF48      Ein Zeichen von Kassette lesen
EF2D:  B0 16     BCS  $EF45      Exit aus Routine: Kassette lesen
EF2F:  48        PHA             Akku Inhalt auf Stack retten
EF30:  20 48 EF  JSR  $EF48      Ein Zeichen von Kassette lesen
EF33:  B0 0D     BCS  $EF42      Fehler aufgetreten, dann Skip
EF35:  D0 05     BNE  $EF3C      Letztes Zeichen vom Band gelesen?
EF37:  A9 40     LDA  # $40      EOF-Kennzeichen in Akku bringen
EF39:  20 57 F7  JSR  $F757      und den Status entsprechend setzen
EF3C:  C6 A6     DEC  * $A6      Bandpuffer Zeiger um 1 vermindern
EF3E:  A6 97     LDX  * $97      X-Reg Inhalt wieder zurückholen
EF40:  68        PLA             Akku Inhalt vom Stack zurückholen
EF41:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   Fehler b. Lesen von Band aufgetreten

EF42:  AA        TAX             Fehlernummer in X-Reg bringen
EF43:  68        PLA             Zeichen zurückholen
EF44:  8A        TXA             Fehlernummer in Akku bringen
EF45:  A6 97     LDX  * $97      X-Reg Inhalt wieder zurückholen
EF47:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   Ein Zeichen von Kassette lesen

EF48:  20 BE E9  JSR  $E9BE      Bandpuffer Zeiger erhöhen
EF4B:  D0 0B     BNE  $EF58      Noch Zeichen im Puffer, dann lesen
EF4D:  20 F2 E9  JSR  $E9F2      Nächsten Block von Kassette lesen
EF50:  B0 09     BCS  $EF5B      STOP Taste gedrückt, dann Abbruch
EF52:  A9 00     LDA  # $00      Akku mit $00 laden und in den Z-Page
```

```
EF54:  85 A6      STA  * $A6     Speicher für Kassettenpuffer Zeiger
EF56:  F0 F0      BEQ  $EF48     Nächstes Zeichen holen
EF58:  B1 B2      LDA  ($B2),Y   Ein Zeichen aus Puffer lesen
EF5A:  18         CLC            Kennzeichen für "OK" setzen
EF5B:  60         RTS            Rücksprung aus dem Unterprogramm

******************************   Zeichen vom IEC Bus holen

EF5C:  A5 90      LDA  * $90     System Status in Akku laden
EF5E:  D0 03      BNE  $EF63     Status nicht "OK", dann Exit
EF60:  4C 3E E4   JMP  $E43E     Kernal ACPTR: Byte v. ser. Bus holen
EF63:  A9 0D      LDA  # $0D     Code für <Cr> in Akku laden
EF65:  18         CLC            Kennzeichen für "OK" setzen
EF66:  60         RTS            Rücksprung aus dem Unterprogramm

******************************   Zeichen von RS-232 holen

EF67:  20 FD EE   JSR  $EEFD     Ein Byte von RS-232 lesen
EF6A:  B0 F9      BCS  $EF65     Fehler aufgetreten, dann Exit
EF6C:  C9 00      CMP  # $00     War gelesenes Zeichen ein Nullbyte?
EF6E:  D0 F6      BNE  $EF66     Nein, dann "OK" Exit
EF70:  AD 14 0A   LDA  $0A14     RS-232 Status in Akku laden
EF73:  29 60      AND  # $60     Fehlt Data set ready (DSR)?
EF75:  D0 EC      BNE  $EF63     Ja, dann <Cr> Code zurückgeben
EF77:  F0 EE      BEQ  $EF67     Nein, dann neuer Leseversuch

******************************   Kernal Routine: BSOUT
                                 Zeichen ausgeben

EF79:  48         PHA            Auszugebendes Zeichen sichern
EF7A:  A5 9A      LDA  * $9A     Hole aktuelles Ausgabegerät
EF7C:  C9 03      CMP  # $03     Ist es der Bildschirm (3)?
EF7E:  D0 04      BNE  $EF84     Nein, dann Skip Bildschirmausgabe
EF80:  68         PLA            Auszugebendes Zeichen zurückholen
EF81:  4C 0C C0   JMP  $C00C     In Routine: Zeichenausgabe Bildschirm

******************************   BSOUT Ausgabe nicht auf Bildschirm

EF84:  90 04      BCC  $EF8A     Auf RS-232 / Datassette ausgeben
EF86:  68         PLA            Auszugebendes Zeichen zurückholen
EF87:  4C 03 E5   JMP  $E503     BSOUT Ausgabe auf IEC (GA größer 3)
EF8A:  4A         LSR  A         Prüfe, ob RS-232 oder Datassette
EF8B:  68         PLA            Auszugebendes Zeichen zurückholen
EF8C:  85 9E      STA  * $9E     und in Z-Page zwischenspeichern
EF8E:  8A         TXA            Den aktuellen Inhalt des X-Reg
EF8F:  48         PHA            über Akku auf Stack retten
EF90:  98         TYA            Den aktuellen Inhalt des Y-Reg
EF91:  48         PHA            über Akku auf Stack retten
```

```
EF92:  90 23     BCC  $EFB7      Sprung in die RS-232 Ausgabe
EF94:  20 BE E9  JSR  $E9BE      Bandpufferzeiger erhöhen
EF97:  D0 0E     BNE  $EFA7      Puffer nicht voll, Zeichen in Puffer
EF99:  20 15 EA  JSR  $EA15      Puffer auf Band schreiben
EF9C:  B0 0E     BCS  $EFAC      Wenn STOP-Taste gedrückt, Abbruch
EF9E:  A9 02     LDA  # $02      Kontrollbyte für Datenblock setzen
EFA0:  A0 00     LDY  # $00      Displacement auf Bandpuffer setzen
EFA2:  91 B2     STA  ($B2),Y    und Kontrollbyte in Puffer schreiben
EFA4:  C8        INY             Das Displacement auf den Bandpuffer
EFA5:  84 A6     STY  * $A6      um 1 erhöhen und in Z-Page sichern
EFA7:  A5 9E     LDA  * $9E      Auszugebendes Zeichen aus Z-Page
EFA9:  91 B2     STA  ($B2),Y    in Ausgabepuffer schreiben
EFAB:  18        CLC             Kennzeichen für OK setzen
EFAC:  68        PLA             Den auf dem Stack gesicherten alten
EFAD:  A8        TAY             Y-Reg Inhalt wiederherstellen
EFAE:  68        PLA             Den auf dem Stack gesicherten alten
EFAF:  AA        TAX             X-Reg Inhalt wiederherstellen
EFB0:  A5 9E     LDA  * $9E      Auszugebendes Zeichen zurückholen
EFB2:  90 02     BCC  $EFB6      Alles OK, dann Rücksprung
EFB4:  A9 00     LDA  # $00      Flag für "STOP" Taste gedrückt
EFB6:  60        RTS             Rücksprung aus dem Unterprogramm

*****************************    RS-232 Zeichen Ausgabe

EFB7:  20 5F E7  JSR  $E75F      Zeichen in RS-232 Puffer schreiben
EFBA:  4C AB EF  JMP  $EFAB      Stack säubern und Rücksprung

*****************************    Kernal Routine: OPEN
                                 Öffnen einer logischen Datei

EFBD:  A6 B8     LDX  * $B8      Logische Dateinummer in X-Reg holen
EFBF:  20 02 F2  JSR  $F202      LFN in LFN-Tabelle suchen
EFC2:  F0 2F     BEQ  $EFF3      Gefunden, dann Fehler ausgeben
EFC4:  A6 98     LDX  * $98      Hole Anzahl der offenen Dateien
EFC6:  E0 0A     CPX  # $0A      Maximal 10 offene sind möglich
EFC8:  B0 26     BCS  $EFF0      Mehr als 10 offen, dann Fehler
EFCA:  E6 98     INC  * $98      Zahl der offenen Dateien +1
EFCC:  A5 B8     LDA  * $B8      Logische Dateinummer in Akku holen
EFCE:  9D 62 03  STA  $0362,X    LFN in LFN-Tabelle eintragen
EFD1:  A5 B9     LDA  * $B9      Sekundäradresse in Akku laden
EFD3:  09 60     ORA  # $60      Print, Input, Get in SA einblenden
EFD5:  85 B9     STA  * $B9      und wieder in SA Speicher merken
EFD7:  9D 76 03  STA  $0376,X    SA in SA-Tabelle eintragen
EFDA:  A5 BA     LDA  * $BA      Geräteadresse in Akku laden
EFDC:  9D 6C 03  STA  $036C,X    GA in GA-Tabelle eintragen
EFDF:  F0 0D     BEQ  $EFEE      War es die Tastatur (0), dann Skip
EFE1:  C9 02     CMP  # $02      Prüfe, ob als Gerät die RS-232
EFE3:  F0 5B     BEQ  $F040      gewählt. Ja, dann Skip zu RS-232
```

```
EFE5:  90 0F     BCC  $EFF6      Kleiner 2, dann ist es Band OPEN
EFE7:  C9 03     CMP  # $03      Prüfe, ob als Gerät der Bildschirm
EFE9:  F0 03     BEQ  $EFEE      gewählt. Ja, dann Skip
EFEB:  20 CB F0  JSR  $F0CB      Datei auf IEC Bus öffnen
EFEE:  18        CLC             Kennzeichen für alles OK setzen
EFEF:  60        RTS             Rücksprung aus dem Unterprogramm

*****************************    OPEN Routine für Bandbetrieb

EFF0:  4C 7C F6  JMP  $F67C      I/O Error #1 (Too many files)
EFF3:  4C 7F F6  JMP  $F67F      I/O Error #2 (File open)
EFF6:  20 80 E9  JSR  $E980      Bandpuffer Startadresse holen
EFF9:  B0 03     BCS  $EFFE      Carry gesetzt, dann gültige Adresse
EFFB:  4C 94 F6  JMP  $F694      I/O Error #9 (Illegal device number)
EFFE:  A5 B9     LDA  * $B9      Sekundäradresse in Akku holen
F000:  29 0F     AND  # $0F      Obere Tetrade (Bit 4-7) ausmaskieren
F002:  D0 1F     BNE  $F023      ungleich Null, auf "Re & Pl" warten
F004:  20 C8 E9  JSR  $E9C8      Wartet auf Taste an Datasette
F007:  B0 36     BCS  $F03F      Ungültig, dann Carry = 1, RTS
F009:  20 0F F5  JSR  $F50F      Steuermeldung "SEARCHING" "FOR"
F00C:  A5 B7     LDA  * $B7      Länge des Dateinamens in Akku
F00E:  F0 0A     BEQ  $F01A      Kein Dateiname vorhanden, dann Skip
F010:  20 9A E9  JSR  $E99A      Suche den entsprechenden Bandheader
F013:  90 18     BCC  $F02D      Wenn gefunden, dann weiter
F015:  F0 28     BEQ  $F03F      Rücksprung mit Carry gesetzt
F017:  4C 85 F6  JMP  $F685      I/O Error #4 (File not found)
F01A:  20 D0 E8  JSR  $E8D0      Nächsten Header auf Kassette suchen
F01D:  90 0E     BCC  $F02D      Wenn gefunden, dann weiter
F01F:  F0 1E     BEQ  $F03F      Rücksprung mit Carry on, da EOT
F021:  B0 F4     BCS  $F017      Weiter suchen, da es PRG File ist
F023:  20 E9 E9  JSR  $E9E9      Warte auf "Record & Play" Taste
F026:  B0 17     BCS  $F03F      Stop Taste gedrückt, dann Abbruch
F028:  A9 04     LDA  # $04      Kontrollcode für Datenheader in Akku
F02A:  20 19 E9  JSR  $E919      Bandheader auf Kassette schreiben
F02D:  A9 BF     LDA  # $BF      Zeiger auf Ende des Bandpuffers in A
F02F:  A4 B9     LDY  * $B9      Sekundäradresse in Y-Reg holen
F031:  C0 60     CPY  # $60      SA Code für Print, Input oder Get?
F033:  F0 07     BEQ  $F03C      Ja, dann Zeiger setzen und RTS
F035:  A0 00     LDY  # $00      Displacement für Bandpuffer setzen
F037:  A9 02     LDA  # $02      Kontrollbyte für Datenblock
F039:  91 B2     STA  ($B2),Y    in den Kassettenpuffer schreiben
F03B:  98        TYA             Displacement von Y nach A kopieren
F03C:  85 A6     STA  * $A6      und Z-Page Bandpufferzeiger setzen
F03E:  18        CLC             Kennzeichen für "OK" setzen
F03F:  60        RTS             Rücksprung aus dem Unterprogramm
```

```
*****************************     RS-232 Open

F040:  20 B0 F0   JSR  $F0B0      CIAs zurücksetzen
F043:  8C 14 0A   STY  $0A14      Z-Page RS-232 Status Byte löschen
F046:  C4 B7      CPY  * $B7      Vgl. mit Länge des Dateinamens
F048:  F0 0B      BEQ  $F055      gleich 0, dann Datenbits berechnen
F04A:  20 AE F7   JSR  $F7AE      1 Byte für RS-232 Register holen
F04D:  99 10 0A   STA  $0A10,Y    RS-232 Kontrollregister,
F050:  C8         INY             Kommandoregister und die
F051:  C0 04      CPY  # $04      Benutzerbaudrate initialisieren
F053:  D0 F1      BNE  $F046      Schleifen,bis 4 Werte übergeben sind
F055:  20 8E E6   JSR  $E68E      Anzahl der Datenbits berechnen
F058:  8E 15 0A   STX  $0A15      Speicher f. Anzahl zu sendender Bits
F05B:  AD 10 0A   LDA  $0A10      RS-232 Kontrollregister laden
F05E:  29 0F      AND  # $0F      Bits für Baudrate isolieren
F060:  F0 1C      BEQ  $F07E      Codewert für Baudrate ermitteln
F062:  0A         ASL  A          Für Tab-Disp. mit 2 multiplizieren
F063:  AA         TAX             Für Index nach X-Reg kopieren
F064:  AD 03 0A   LDA  $0A03      Hole PAL/NTSC Zeiger
F067:  D0 09      BNE  $F072      Nicht NTSC Version, dann Skip
F069:  BC 4F E8   LDY  $E84F,X    Timerkonstante RS-232 B-Rate NTSC Hi
F06C:  BD 4E E8   LDA  $E84E,X    Timerkonstante RS-232 B-Rate NTSC Lo
F06F:  4C 78 F0   JMP  $F078      Skip zur Sicherung der Baudrate
F072:  BC 63 E8   LDY  $E863,X    Timerkonstante RS-232 B-Rate PAL Hi
F075:  BD 62 E8   LDA  $E862,X    Timerkonstante RS-232 B-Rate PAL Lo
F078:  8C 13 0A   STY  $0A13      Hi Wert der Baudrate sichern
F07B:  8D 12 0A   STA  $0A12      Lo Wert der Baudrate sichern
F07E:  AD 12 0A   LDA  $0A12      Lo-Wert der Baudrate aus Speicher
F081:  0A         ASL  A          holen und mit 2 multiplizieren
F082:  AA         TAX             Wert im X-Reg sichern
F083:  AD 13 0A   LDA  $0A13      Hi Wert der Baudrate aus Speicher
F086:  2A         ROL  A          holen und mit 2 multiplizieren
F087:  A8         TAY             Wert im Y-Reg sichern
F088:  8A         TXA             Lo-Wert für Codeermittlung in Akku
F089:  69 C8      ADC  # $C8      Dezimal 200 addieren
F08B:  8D 16 0A   STA  $0A16      Timerwert f. Sende-Baudrate sichern
F08E:  98         TYA             Hi-Wert für Codeermittlung in Akku
F08F:  69 00      ADC  # $00      Dezimal 000 addieren
F091:  8D 17 0A   STA  $0A17      Timerwert f. Sende-Baudrate sichern
F094:  AD 11 0A   LDA  $0A11      RS-232 Kommandoregister holen
F097:  4A         LSR  A          Prüfe auf 3-Line Handshake
F098:  90 09      BCC  $F0A3      Ja, dann Skip DSR Test
F09A:  AD 01 DD   LDA  $DD01      Prüfe, ob das DATA SET READY
F09D:  0A         ASL  A          (DSR) Signal fehlt
F09E:  B0 03      BCS  $F0A3      Nein, dann Skip
F0A0:  20 55 E7   JSR  $E755      Status für DSR setzen
F0A3:  AD 18 0A   LDA  $0A18      Anfang des RS-232 Eingabepuffers mit
F0A6:  8D 19 0A   STA  $0A19      Ende des Eingabepuffers gleichsetzen
```

```
F0A9:  AD 1B 0A    LDA  $0A1B      Anfang des RS-232 Ausgabepuffers mit
F0AC:  8D 1A 0A    STA  $0A1A      Ende des Ausgabepuffers gleichsetzen
F0AF:  60          RTS             Rücksprung aus dem Unterprogramm

******************************     CIAs nach RS-232 zurücksetzen

F0B0:  A9 7F       LDA  # $7F      Wert für "Interrupt löschen" in Akku
F0B2:  8D 0D DD    STA  $DD0D      IRQs zurücksetzen
F0B5:  A9 06       LDA  # $06      Bits 1 und 2 auf Ausgang setzen
F0B7:  8D 03 DD    STA  $DD03      Datenrichtungsregister Port B
F0BA:  8D 01 DD    STA  $DD01      Port Register Port B
F0BD:  A9 04       LDA  # $04      Bit 2 des Datenport A (CIA 2)
F0BF:  0D 00 DD    ORA  $DD00      für die RS-232 Datenausgabe
F0C2:  8D 00 DD    STA  $DD00      einblenden (TXD Signal)
F0C5:  A0 00       LDY  # $00      Y-Reg mit $00 vorladen und das
F0C7:  8C 0F 0A    STY  $0A0F      RS-232 NMI Flag löschen
F0CA:  60          RTS             Rücksprung aus dem Unterprogramm

******************************     Datei auf IEC Bus öffnen

F0CB:  A5 B9       LDA  * $B9      Sekundäradresse in Akku laden
F0CD:  30 04       BMI  $F0D3      Wenn Bit 7 für "CLOSE" gesetzt, Exit
F0CF:  A4 B7       LDY  * $B7      Hole Länge des Dateinamens
F0D1:  D0 02       BNE  $F0D5      Ungleich Null, dann weiter
F0D3:  18          CLC             Carry für Kennzeichen "OK" löschen
F0D4:  60          RTS             Rücksprung aus dem Unterprogramm

******************************     Dateiname an seriellen Bus senden

F0D5:  A9 00       LDA  # $00      Setze das Statusbyte auf das
F0D7:  85 90       STA  * $90      Kennzeichen $00 (= alles OK)
F0D9:  A5 BA       LDA  * $BA      Geräteadresse in Akku laden
F0DB:  20 3E E3    JSR  $E33E      Ende RS-232 Übertragung abwarten
F0DE:  24 90       BIT  * $90      STATUS auf gesetztes EOF Bit testen
F0E0:  30 0B       BMI  $F0ED      Wenn EOF, dann Fehler ausgeben
F0E2:  A5 B9       LDA  * $B9      Sekundäradresse in Akku laden
F0E4:  09 F0       ORA  # $F0      Steuertetrade in SA einblenden
F0E6:  20 D2 E4    JSR  $E4D2      Routine SECND: Sekundäradr. f. LISTN
F0E9:  A5 90       LDA  * $90      System STATUS in Akku laden
F0EB:  10 05       BPL  $F0F2      Wenn OK, dann normal weiter
F0ED:  68          PLA             RTS Adresse vom Stack löschen
F0EE:  68          PLA             RTS Adresse vom Stack löschen
F0EF:  4C 88 F6    JMP  $F688      I/O Error #5 (Device not present)
F0F2:  A5 B7       LDA  * $B7      Hole Länge des Dateinamens
F0F4:  F0 0D       BEQ  $F103      Kein Name angegeben, dann Skip
F0F6:  A0 00       LDY  # $00      Displ. auf 1. Zeichen d. Dateinamens
F0F8:  20 AE F7    JSR  $F7AE      1 Zeichen d. Dateinamens lesen
F0FB:  20 03 E5    JSR  $E503      Kernal CIOUT: Byte an seriellen Bus
```

```
F0FE:  C8        INY             Displacementzeiger um 1 erhöhen
F0FF:  C4 B7     CPY  * $B7      Displacement = Dateinamenlänge?
F101:  D0 F5     BNE  $F0F8      Nein, dann weiter ausgeben
F103:  4C B0 F5  JMP  $F5B0      UNLSN an seriellen Bus und RTS

******************************   Kernal Routine: CHKIN
                                 Eingabekanal setzen

F106:  20 02 F2  JSR  $F202      LFN in LFN-Tabelle suchen
F109:  D0 3E     BNE  $F149      I/O Error #3 (File not found)
F10B:  20 12 F2  JSR  $F212      LFN,GA,SA entsprechend neu setzen
F10E:  F0 13     BEQ  $F123      GA = 0, dann Standard setzen
F110:  C9 03     CMP  # $03      Ist es die GA 3 (= Bildschirm )?
F112:  F0 0F     BEQ  $F123      Ja,dann Bildschirm f.Standard setzen
F114:  B0 11     BCS  $F127      Größer 3, dann IEC Auswertung
F116:  C9 02     CMP  # $02      Prüfe, ob RS-232 gewählt wurde
F118:  D0 03     BNE  $F11D      Nein, dann war es die Datasette
F11A:  4C 95 E7  JMP  $E795      Zur RS-232 Eingabe
F11D:  A6 B9     LDX  * $B9      Sekundäradresse in X-Reg holen
F11F:  E0 60     CPX  # $60      Ist die Sekundäradresse = 0?
F121:  D0 20     BNE  $F143      I/O Error #6 (Not input file)
F123:  85 99     STA  * $99      In Z-Page f. Standard Eingabegerät
F125:  18        CLC             Kennzeichen für "OK" setzen
F126:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   Auswertung bei CHKIN auf IEC

F127:  AA        TAX             Sichere Geräteadresse im X-Reg
F128:  20 3B E3  JSR  $E33B      Routine TALK: TALK Befehl an ser.Bus
F12B:  24 90     BIT  * $90      STATUS auf gesetztes EOF-Bit testen
F12D:  30 11     BMI  $F140      Bit 7 gesetzt = "Device not present"
F12F:  A5 B9     LDA  * $B9      Sekundäradresse in Akku laden
F131:  10 05     BPL  $F138      Sekundäradresse für TALK senden
F133:  20 E9 E4  JSR  $E4E9      Wartet auf Taktsignal
F136:  10 03     BPL  $F13B      Skip Ausgabe d. TALK Sekundäradresse
F138:  20 E0 E4  JSR  $E4E0      Routine TKSA: Sekundäradr. für TALK
F13B:  8A        TXA             Geräteadresse wieder in Akku zurück
F13C:  24 90     BIT  * $90      STATUS auf gesetztes EOF-Bit testen
F13E:  10 E3     BPL  $F123      Alles OK,dann a. Eingabegerät setzen
F140:  4C 88 F6  JMP  $F688      I/O Error #5 (Device not present)
F143:  4C 8B F6  JMP  $F68B      I/O Error #6 (Not input file)
F146:  4C 8E F6  JMP  $F68E      I/O Error #7 (Not output file)
F149:  4C 82 F6  JMP  $F682      I/O Error #3 (File not open)

******************************   Kernal Routine: CKOUT
                                 Ausgabekanal setzen

F14C:  20 02 F2  JSR  $F202      LFN in LFN-Tabelle suchen
```

```
F14F:  D0 F8     BNE  $F149      I/O Error #3 (File not open)
F151:  20 12 F2  JSR  $F212      LFN,GA,SA entsprechend neu setzen
F154:  F0 F0     BEQ  $F146      I/O Error #7 (Not output file)
F156:  C9 03     CMP  # $03      Vgl mit GA 3 (= Bildschirm)
F158:  F0 0F     BEQ  $F169      Ja, dann als Standard Ausgabe setzen
F15A:  B0 11     BCS  $F16D      GA größer 3, dann IEC Auswertung
F15C:  C9 02     CMP  # $02      Prüfe, ob RS-232 gewählt wurde
F15E:  D0 03     BNE  $F163      Nein, dann Skip
F160:  4C 29 E7  JMP  $E729      Zur RS-232 Ausgabe
F163:  A6 B9     LDX  * $B9      Sekundäradresse in X-Reg holen
F165:  E0 60     CPX  # $60      Ist die Sekundäradresse = 0?
F167:  F0 DD     BEQ  $F146      I/O Error #7 (Not output file)
F169:  85 9A     STA  * $9A      In Z-Page für Standard Ausgabegerät
F16B:  18        CLC             Kennzeichen für "OK" setzen
F16C:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   Auswertung bei CKOUT auf IEC

F16D:  AA        TAX             Geräteadresse für LISTN in X-Reg
F16E:  20 3E E3  JSR  $E33E      Routine LISTN: LISTN Bef.an ser. Bus
F171:  24 90     BIT  * $90      STATUS auf gesetztes EOF Bit testen
F173:  30 CB     BMI  $F140      I/O Error #5 (Device not present)
F175:  A5 B9     LDA  * $B9      Sekundäradresse in Akku laden
F177:  10 05     BPL  $F17E      OPEN/CLOSE Bit clear, dann Skip
F179:  20 D7 E4  JSR  $E4D7      ATN Signal zurücksetzen
F17C:  D0 03     BNE  $F181      Skip Ausgabe der LISTN Sekundäradr.
F17E:  20 D2 E4  JSR  $E4D2      Routine SECND: Sekundäradr. f. LISTN
F181:  8A        TXA             Geräteadresse wieder in Akku zurück
F182:  24 90     BIT  * $90      STATUS auf gesetztes EOF Bit testen
F184:  10 E3     BPL  $F169      Alles OK, dann entsprechenden RTS
F186:  30 B8     BMI  $F140      I/O Error #5 (Device not present)

******************************   Kernal Routine: CLOSE
                                 Eine Datei schließen

F188:  66 92     ROR  * $92      Carry als KZ in Z-Page Flag rotieren
F18A:  20 07 F2  JSR  $F207      LFN in LFN-Tabelle suchen
F18D:  D0 DC     BNE  $F16B      Nicht gefunden, dann "OK" Rücksprung
F18F:  20 12 F2  JSR  $F212      LFN,GA,SA entspr. Tabelle erneuern
F192:  8A        TXA             Tabellen Displacementzeiger
F193:  48        PHA             auf dem Stack sichern
F194:  A5 BA     LDA  * $BA      Geräteadresse in Akku laden
F196:  F0 4C     BEQ  $F1E4      War angesprochenes Gerät Tastatur?
F198:  C9 03     CMP  # $03      Prüfe, ob angesprochenes Gerät der
F19A:  F0 48     BEQ  $F1E4      Bildschirm (3) war. Ja, dann Skip
F19C:  B0 31     BCS  $F1CF      War es ein Gerät am IEC Bus?
F19E:  C9 02     CMP  # $02      War es die RS-232?
F1A0:  D0 07     BNE  $F1A9      Nein, dann Close an Kassette
```

```
F1A2:  68        PLA              Das Displ. auf d. Tab. zurückholen
F1A3:  20 E5 F1  JSR  $F1E5       Dateieintrag aus Tabelle löschen
F1A6:  4C B0 F0  JMP  $F0B0       CIA's zurücksetzen und RTS

*****************************     Schließen einer Band-Datei

F1A9:  A5 B9     LDA  * $B9       Sekundäradresse in Akku laden
F1AB:  29 0F     AND  # $0F       Obere Tetrade (Bit 4-7) ausmaskieren
F1AD:  F0 35     BEQ  $F1E4       Dateieintrag aus Tabelle löschen
F1AF:  20 80 E9  JSR  $E980       Bandpuffer Adresse holen und prüfen
F1B2:  A9 00     LDA  # $00       Kennzeichen für Close setzen und
F1B4:  38        SEC              Steuerkennzeichen Carry setzen
F1B5:  20 8C EF  JSR  $EF8C       Zeichen in Puffer schreiben
F1B8:  20 15 EA  JSR  $EA15       Puffer auf Band schreiben
F1BB:  90 04     BCC  $F1C1       Alles Ok, dann weiter mit Tape Close
F1BD:  68        PLA              Auszugebendes Zeichen zurückholen
F1BE:  A9 00     LDA  # $00       und durch CHR$(0) ersetzen
F1C0:  60        RTS              Rücksprung aus dem Unterprogramm

*****************************     Löschen eines Dateieintrages

F1C1:  A5 B9     LDA  * $B9       Sekundäradresse in Akku laden
F1C3:  C9 62     CMP  # $62       War die untere Tetrade der SA = 2?
F1C5:  D0 1D     BNE  $F1E4       Dateieintrag aus Tabelle löschen
F1C7:  A9 05     LDA  # $05       Kontrollbyte für EOT Header setzen
F1C9:  20 19 E9  JSR  $E919       und Datenblock auf Band schreiben
F1CC:  4C E4 F1  JMP  $F1E4       Dateieintrag aus Tabelle löschen
F1CF:  24 92     BIT  * $92       Prüfe Tape Zeitkonstante
F1D1:  10 0E     BPL  $F1E1       Kleiner 128, dann Close senden
F1D3:  A5 BA     LDA  * $BA       Geräteadresse in Akku laden
F1D5:  C9 08     CMP  # $08       War es ein Diskettenlaufwerk (8-15)
F1D7:  90 08     BCC  $F1E1       Nein, dann Skip Floppy Close
F1D9:  A5 B9     LDA  * $B9       Sekundäradresse in Akku laden
F1DB:  29 0F     AND  # $0F       Obere Tetrade (Bit 4-7) ausmaskieren
F1DD:  C9 0F     CMP  # $0F       War Befehlskanal (15) geöffnet, dann
F1DF:  F0 03     BEQ  $F1E4       Dateieintrag aus Tabelle löschen
F1E1:  20 9E F5  JSR  $F59E       CLOSE Befehl an Gerät schicken

*****************************     Dateieintrag aus Tabelle löschen

F1E4:  68        PLA              Displ. auf Tabelle zurückholen
F1E5:  AA        TAX              Displ. von A in X-Reg kopieren
F1E6:  C6 98     DEC  * $98       Zahl der offenen Dateien - 1
F1E8:  E4 98     CPX  * $98       War der gefundene Tabelleneintrag
F1EA:  F0 14     BEQ  $F200       d. letzte Tabelleneintrag, dann Exit
F1EC:  A4 98     LDY  * $98       Zahl offener Dateien f. Displ. holen
F1EE:  B9 62 03  LDA  $0362,Y     Letzten Eintrag aus LFN-Tab. holen
F1F1:  9D 62 03  STA  $0362,X     und an freie Position kopieren
```

```
F1F4:  B9 6C 03   LDA  $036C,Y    Letzten Eintrag aus GA-Tab. holen
F1F7:  9D 6C 03   STA  $036C,X    und an freie Position kopieren
F1FA:  B9 76 03   LDA  $0376,Y    Letzten Eintrag aus SA-Tab. holen
F1FD:  9D 76 03   STA  $0376,X    und an freie Position kopieren
F200:  18         CLC             Kennzeichen für "OK" setzen
F201:  60         RTS             Rücksprung aus dem Unterprogramm

*****************************     LFN in X-Reg in LFN Tabelle suchen

F202:  A9 00      LDA  # $00      Statusbyte löschen und somit
F204:  85 90      STA  * $90      auf Kennzeichen: "Alles OK" setzen
F206:  8A         TXA             Suchwert für LFN in Akku kopieren
F207:  A6 98      LDX  * $98      Zahl der offenen Dateien holen
F209:  CA         DEX             Um 1 vermindern,da als Index benutzt
F20A:  30 05      BMI  $F211      Alle Vergleiche negativ, dann Exit
F20C:  DD 62 03   CMP  $0362,X    Mit Byte aus LFN-Tabelle vergleichen
F20F:  D0 F8      BNE  $F209      Nicht gleich, dann nächsten Vgl.
F211:  60         RTS             Rücksprung aus dem Unterprogramm

*****************************     LFN,GA,SA entsprechend dem X-Reg
                                  Displacement aus Tabellen holen

F212:  BD 62 03   LDA  $0362,X    Die durch X-Reg Displ. bestimmte
F215:  85 B8      STA  * $B8      log. Dateinr. in Z-Page Byte f. LFN
F217:  BD 76 03   LDA  $0376,X    Die durch X-Reg Displ. bestimmte
F21A:  85 B9      STA  * $B9      Sekundäradr. in Z-Page Byte für SA
F21C:  BD 6C 03   LDA  $036C,X    Die durch X-Reg Displ. bestimmte
F21F:  85 BA      STA  * $BA      Geräteadr. in Z-Page Byte für GA
F221:  60         RTS             Rücksprung aus dem Unterprogramm

*****************************     Kernal Routine: CLALL
                                  Alle offenen Dateien zurücksetzen

F222:  A9 00      LDA  # $00      Akku mit 0 laden und in Z-Page
F224:  85 98      STA  * $98      Speicher f. Anzahl d. offenen Dat.

*****************************     Kernal Routine: CLRCH
                                  Ein-/Ausgabekanal zurücksetzen

F226:  A2 03      LDX  # $03      Code für Gerät Bildschirm (3) laden
F228:  E4 9A      CPX  * $9A      Mit aktuellem Ausgabegerät vgl.
F22A:  B0 03      BCS  $F22F      In CLRCH Rout. für Gerät am IEC Bus
F22C:  20 26 E5   JSR  $E526      Routine UNLSN: UNLSN Bef.an ser. Bus
F22F:  E4 99      CPX  * $99      Mit aktuellem Eingabegerat vgl.
F231:  B0 03      BCS  $F236      In CLRCH Rout. für Gerät am IEC Bus
F233:  20 15 E5   JSR  $E515      Routine UNTLK: UNTLK Bef.an ser. Bus
F236:  86 9A      STX  * $9A      Als Ausgabegerät den Bildschirm und
F238:  A9 00      LDA  # $00      als Standard Eingabegerät wieder
```

```
F23A:  85 99      STA  * $99      die Tastatur setzen.
F23C:  60         RTS             Rücksprung aus dem Unterprogramm

******************************    Standard E-/A- Geräte setzen

F23D:  85 BA      STA  * $BA      In Z-Page Byte f.aktuelle Geräteadr.
F23F:  C5 9A      CMP  * $9A      Mit aktuellem Ausgabegerät vgl.
F241:  D0 05      BNE  $F248      Nicht gleich, mit Eingabegerät vgl.
F243:  A9 03      LDA  # $03      Akku mit GA für Bildschirm laden (3)
F245:  85 9A      STA  * $9A      und als Ausgabegerät setzen
F247:  2C         .Byte  $2C      Skip nach $F24A
F24(:  C5 99      CMP  # $99      Mit aktuellem Eingabegerät vgl.
F24A:  D0 04      BNE  $F250      Nicht gleich, dann suche in GA-Tab
F24C:  A9 00      LDA  # $00      Akku mit Code für Tastatur laden (0)
F24E:  85 99      STA  * $99      u. als Eingabe d. Tastatur setzen
F250:  A5 BA      LDA  * $BA      Geräteadresse in Akku laden
F252:  A6 98      LDX  * $98      Zahl d. offenen Dateien in X-Reg
F254:  CA         DEX             Um 1 vermindern,da als Index benutzt
F255:  30 0D      BMI  $F264      Alle Vergleiche negativ, dann Exit
F257:  DD 6C 03   CMP  $036C,X    Mit Tabelle für Geräteadressen vgl.
F25A:  D0 F8      BNE  $F254      Nicht gefunden, dann nächster Vgl.
F25C:  BD 62 03   LDA  $0362,X    LFN zur entspr. GA holen
F25F:  20 C3 FF   JSR  $FFC3      Kernal CLOSE: Datei schließen
F262:  90 EC      BCC  $F250      Wenn Carry Clear, nächster Close
F264:  60         RTS             Rücksprung aus dem Unterprogramm

******************************    Kernal Routine: LOAD
                                  Lade Datei in einen Speicherbereich

F265:  86 C3      STX  * $C3      Startadresse Lo in Z-Page ablegen
F267:  84 C4      STY  * $C4      Startadresse Hi in Z-Page ablegen
F269:  6C 30 03   JMP  ($0330)    Vektor zeigt auf LOAD Rout.($F26C)
F26C:  85 93      STA  * $93      Z-Page Flag für LOAD/VERIFY
F26E:  A9 00      LDA  # $00      Akku mit $00 laden und
F270:  85 90      STA  * $90      Status auf "alles OK" setzen
F272:  A5 BA      LDA  * $BA      Geräteadresse in Akku laden
F274:  C9 04      CMP  # $04      Prüfe auf gültige Geräteadresse
F276:  B0 03      BCS  $F27B      Geräteadresse größer 4 ist gültig
F278:  4C 26 F3   JMP  $F326      Prüfe auf Datasette, sonst ungültig

******************************    LOAD Routine vom IEC-Bus

F27B:  AD 1C 0A   LDA  $0A1C      Systemzeiger für schnellen seriellen
F27E:  29 BE      AND  # $BE      Modus auslesen und Bit 6 (1 = fast,
F280:  8D 1C 0A   STA  $0A1C      0 = slow) eliminieren
F283:  A6 B9      LDX  * $B9      Sekundäradresse in X-Reg holen
F285:  86 9E      STX  * $9E      und in Z-Page $9E zwischenspeichern
F287:  A4 B7      LDY  * $B7      Hole Länge des Dateinamens
```

```
F289:  D0 03     BNE  $F28E     Ungleich 0, dann Skip Fehlermeldung
F28B:  4C 1A F3  JMP  $F31A     I/O Error #8 (Missing filename)
F28E:  84 9F     STY  * $9F     Länge Dateinamen zwischenspeichern
F290:  20 0F F5  JSR  $F50F     "Searching for" Meldung ausgeben
F293:  20 A1 F3  JSR  $F3A1     Prüfe Dateinamen und Fast Ser. Mode
F296:  B0 03     BCS  $F29B     Carry gesetzt, dann Fehler. Skip
F298:  4C 9B F3  JMP  $F39B     Load-Endadresse setzen und RTS
F29B:  A4 9F     LDY  * $9F     Länge des Dateinamens in Y-Reg und
F29D:  84 B7     STY  * $B7     in Z-Page für Länge des Dateinamens
F29F:  A9 60     LDA  # $60     SA 0, Hi-Tetrade für Input/Get
F2A1:  85 B9     STA  * $B9     in Z-Page für Sekundäradresse
F2A3:  20 CB F0  JSR  $F0CB     TALK Befehl an seriellen Bus senden
F2A6:  A5 BA     LDA  * $BA     Geräteadresse in Akku laden
F2A8:  20 3B E3  JSR  $E33B     Routine TALK: TALK Befehl an ser.Bus
F2AB:  A5 B9     LDA  * $B9     Sekundäradresse in Akku laden
F2AD:  20 E0 E4  JSR  $E4E0     Routine TKSA: Sekundäradr. für TALK
F2B0:  20 3E E4  JSR  $E43E     Ein Byte vom IEC Bus holen
F2B3:  85 AE     STA  * $AE     Startadresse Lo in Z-Page ablegen
F2B5:  20 3E E4  JSR  $E43E     Ein Byte vom IEC Bus holen
F2B8:  85 AF     STA  * $AF     Startadresse Hi in Z-Page ablegen
F2BA:  A5 90     LDA  * $90     System STATUS in Akku laden
F2BC:  4A        LSR  A         Timeout Bit rechts shiften
F2BD:  4A        LSR  A         Timeout Bit ins Carry Flag schieben
F2BE:  B0 57     BCS  $F317     Timeout beim Lesen, (File not found)
F2C0:  A5 9E     LDA  * $9E     Hole gesicherte Sekundäradresse
F2C2:  D0 08     BNE  $F2CC     Ungleich 0, dann Skip
F2C4:  A5 C3     LDA  * $C3     Die durch X-Reg und Y-Reg
F2C6:  85 AE     STA  * $AE     übergebene Startadresse für
F2C8:  A5 C4     LDA  * $C4     den LOAD Befehl von $C3,$C4
F2CA:  85 AF     STA  * $AF     nach $AE,$AF kopieren
F2CC:  20 33 F5  JSR  $F533     Steuermeldung a. Bildschirm anzeigen
F2CF:  A9 FD     LDA  # $FD     Das Statusbit für Timeout beim Lesen
F2D1:  25 90     AND  * $90     aus dem Status ausblenden und
F2D3:  85 90     STA  * $90     wieder in den Status zurückschreiben
F2D5:  20 E1 FF  JSR  $FFE1     Kernal STOP: Auf Stop Taste prüfen
F2D8:  F0 49     BEQ  $F323     Zum Abbruch der LOAD Routine
F2DA:  20 3E E4  JSR  $E43E     Kernal Routine: ACPTR
F2DD:  AA        TAX            Akku Inhalt im X-Reg sichern
F2DE:  A5 90     LDA  * $90     System STATUS in Akku laden
F2E0:  4A        LSR  A         Das "Timeout beim Lesen" Bit aus
F2E1:  4A        LSR  A         dem Statusbyte eliminieren
F2E2:  B0 EB     BCS  $F2CF     Wenn Timeout, dann neuer Leseversuch
F2E4:  8A        TXA            Alten Akku-Inhalt wiederherstellen
F2E5:  A4 93     LDY  * $93     Teste Z-Page LOAD/VERIFY Zeiger
F2E7:  F0 12     BEQ  $F2FB     Wenn Null, dann ist es LOAD
F2E9:  85 BD     STA  * $BD     In Z-Page Paritätspuffer ablegen
F2EB:  A0 00     LDY  # $00     Displacementzeiger für FETCH Routine
F2ED:  20 C9 F7  JSR  $F7C9     FETCH Routine für LSV Operationen
```

```
F2F0:  C5 BD     CMP  * $BD     Vergleiche m. Z-Page Paritäts-Puffer
F2F2:  F0 0A     BEQ  $F2FE     Wenn gleich, dann OK und Skip
F2F4:  A9 10     LDA  # $10     Bit 4 im System Status einblenden
F2F6:  20 57 F7  JSR  $F757     Kernal STATUS: System Status setzen
F2F9:  D0 03     BNE  $F2FE     Nicht OK, dann Skip abspeichern
F2FB:  20 BF F7  JSR  $F7BF     INDSTA Routine über Z-Page $AE-$AF
F2FE:  E6 AE     INC  * $AE     Lo-Byte des Speicherzeigers +1
F300:  D0 08     BNE  $F30A     Kein Überlauf, dann Skip Abgleich
F302:  E6 AF     INC  * $AF     Hi-Byte des Speicherzeigers +1
F304:  A5 AF     LDA  * $AF     Prüfe, ob Hi-Byte in $FF00 Bereich
F306:  C9 FF     CMP  # $FF     zeigt. Wenn ja, dann springe in
F308:  F0 16     BEQ  $F320     die Fehlerausgabe
F30A:  24 90     BIT  * $90     STATUS auf gesetztes EOF Bit testen
F30C:  50 C1     BVC  $F2CF     Noch kein EOF, dann weiter
F30E:  20 15 E5  JSR  $E515     Routine UNTLK: UNTLK Bef.an ser. Bus
F311:  20 9E F5  JSR  $F59E     UNLSTN und CLOSE an ser. Bus senden
F314:  4C 9B F3  JMP  $F39B     Carry löschen und Rücksprung
F317:  4C 85 F6  JMP  $F685     I/O Error #4 (File not found)
F31A:  4C 91 F6  JMP  $F691     I/O Error #8 (Missing filename)
F31D:  4C 94 F6  JMP  $F694     I/O Error #9 (Illegal device number)
F320:  4C 97 F6  JMP  $F697     I/O Error #0
F323:  4C B5 F5  JMP  $F5B5     Sprung,wenn LOAD Routine abgebrochen
F326:  C9 01     CMP  # $01     Ist es ein LOAD von Datasette?
F328:  D0 F3     BNE  $F31D     Nein, dann I/O Error #9
F32A:  20 80 E9  JSR  $E980     Hole und prüfe Bandpuffer Adresse
F32D:  90 EE     BCC  $F31D     Bandpuffer Adresse ungültig, Fehler
F32F:  20 C8 E9  JSR  $E9C8     Warte auf Taste am Kassettenrekorder
F332:  B0 6C     BCS  $F3A0     Abbruch durch STOP Taste, dann RTS
F334:  20 0F F5  JSR  $F50F     "SEARCHING" "FOR" Dateiname ausgeben
F337:  A5 B7     LDA  * $B7     Z-Page Speicher für Dateinamenlänge
F339:  F0 09     BEQ  $F344     Länge = 0, Skip nach Namen suchen
F33B:  20 9A E9  JSR  $E99A     Bandheader nach Namen suchen
F33E:  90 0B     BCC  $F34B     OK, dann weiter
F340:  F0 5E     BEQ  $F3A0     Abbruch durch STOP Taste, dann RTS
F342:  B0 D3     BCS  $F317     I/O Error #4 (File not found)
F344:  20 D0 E8  JSR  $E8D0     Programm Header vom Band lesen
F347:  F0 57     BEQ  $F3A0     Abbruch durch STOP Taste, dann RTS
F349:  B0 CC     BCS  $F317     I/O Error #4 (File not found)
F34B:  38        SEC            Kennzeichen: Fehler gefunden setzen
F34C:  A5 90     LDA  * $90     System STATUS in Akku laden
F34E:  29 10     AND  # $10     Bit 4 für Lesefehler eliminieren
F350:  D0 4E     BNE  $F3A0     Bit 4 (Lesefehler) gesetzt, dann RTS
F352:  E0 01     CPX  # $01     Code für Header Typ #1 (BASIC-Prg)
F354:  F0 11     BEQ  $F367     Ist es ein BASIC-Programm, Skip
F356:  E0 03     CPX  # $03     Code für Header Typ #3 (Masch. Prg)
F358:  D0 DD     BNE  $F337     Wenn nicht #1 oder #3, weiter suchen
F35A:  A0 01     LDY  # $01     Displacement auf Kassettenpuffer
F35C:  B1 B2     LDA  ($B2),Y   Hole Startadresse Lo aus Puffer und
```

```
F35E:  85 C3     STA  * $C3     kopiere sie in Ladeadreßzeiger Lo
F360:  C8        INY            Displacement im Kassettenpuffer +1
F361:  B1 B2     LDA  ($B2),Y   Hole Startadresse Hi aus Puffer und
F363:  85 C4     STA  * $C4     kopiere sie in Ladeadreßzeiger Hi
F365:  B0 04     BCS  $F36B     Unbedingter Sprung bei Masch. Prg.
F367:  A5 B9     LDA  * $B9     Sekundäradresse in Akku laden
F369:  D0 EF     BNE  $F35A     Ist sie 0 (Append)? Nein, dann Skip
F36B:  A0 03     LDY  # $03     Displacement auf Kassettenpuffer
F36D:  B1 B2     LDA  ($B2),Y   Hole Endadresse Lo aus Puffer
F36F:  A0 01     LDY  # $01     Displacement auf Kassettenpuffer
F371:  F1 B2     SBC  ($B2),Y   Subtr. Startadresse Lo v. Endadresse
F373:  AA        TAX            und sichere Lo Wert im X-Register
F374:  A0 04     LDY  # $04     Displacement auf Kassettenpuffer
F376:  B1 B2     LDA  ($B2),Y   Hole Endadresse Hi aus Puffer
F378:  A0 02     LDY  # $02     Displacement auf Kassettenpuffer
F37A:  F1 B2     SBC  ($B2),Y   Subtr. Startadresse Hi v. Endadresse
F37C:  A8        TAY            und sichere Hi Wert im Y-Register
F37D:  18        CLC            Carry für Addition löschen
F37E:  8A        TXA            Programmlänge Lo in Akku zurück
F37F:  65 C3     ADC  * $C3     Speicherstartadresse + Programmlänge
F381:  85 AE     STA  * $AE     in Zeiger für Endadresse Lo ablegen
F383:  98        TYA            Programmlänge Hi in Akku zurück
F384:  65 C4     ADC  * $C4     Speicherstartadresse + Programmlänge
F386:  85 AF     STA  * $AF     in Zeiger für Endadresse Hi ablegen
F388:  C9 FF     CMP  # $FF     Reicht Endadresse in $FF00 Bereich?
F38A:  F0 94     BEQ  $F320     Ja, dann I/O Error #0
F38C:  A5 C3     LDA  * $C3     Die Speicherstartadresse Lo in den
F38E:  85 C1     STA  * $C1     Z-Page Ladezeiger Lo kopieren
F390:  A5 C4     LDA  * $C4     Die Speicherstartadresse Hi in den
F392:  85 C2     STA  * $C2     Z-Page Ladezeiger Hi kopieren
F394:  20 33 F5  JSR  $F533     "LOADING"/"VERIFYING" Text ausgeben
F397:  20 FB E9  JSR  $E9FB     Programm vom Band laden
F39A:  24        .Byte  $24     Skip nach $F39C

******************************  Programmendadresse nach LOAD setzen

F39B:  18        CLC            Carry für Kennzeichen OK setzen
F39C:  A6 AE     LDX  * $AE     Programmendadresse Lo in X-Reg
F39E:  A4 AF     LDY  * $AF     Programmendadresse Hi in Y-Reg
F3A0:  60        RTS            Rücksprung aus dem Unterprogramm

******************************  Prüfe Dateinamen und den
                                "Fast Seriell Mode"

F3A1:  A0 00     LDY  # $00     Displ. für FETCH Routine setzen
F3A3:  20 AE F7  JSR  $F7AE     Byte des Dateinam. a.bel. Bank holen
F3A6:  C9 24     CMP  # $24     Ist erstes Zeichen ein <$>?
F3A8:  F0 F6     BEQ  $F3A0     Ja, dann Rücksprung: RTS
```

```
F3AA:  A6 BA     LDX  * $BA     Geräteadresse in X-Reg laden
F3AC:  A0 0F     LDY  # $0F     Sekundäradresse auf (15) setzen
F3AE:  A9 00     LDA  # $00     Logische Dateinummer auf (0) setzen
F3B0:  20 38 F7  JSR  $F738     Routine SETLFS: Setze Dateiparameter
F3B3:  85 B7     STA  * $B7     Länge des Dateinamens auf 0 setzen
F3B5:  20 C0 FF  JSR  $FFC0     Kernal OPEN: Datei öffnen
F3B8:  A6 B8     LDX  * $B8     Logische Dateinummer in X-Reg holen
F3BA:  20 C9 FF  JSR  $FFC9     Kernal CKOUT: Ausgabekanal setzen
F3BD:  90 08     BCC  $F3C7     Kein Fehler, dann weiter
F3BF:  20 8C F4  JSR  $F48C     Logische Datei wieder schließen
F3C2:  68        PLA            Die RTS Adresse vom Stack entfernen
F3C3:  68        PLA            Die RTS Adresse vom Stack entfernen
F3C4:  4C 88 F6  JMP  $F688     I/O Error #5 (Device not present)
F3C7:  A0 03     LDY  # $03     Schleifen und Displacementzähler
F3C9:  B9 0B F5  LDA  $F50B,Y   Befehlssequenzstring für Floppy
F3CC:  20 D2 FF  JSR  $FFD2     Kernal BSOUT: Ein Zeichen ausgeben
F3CF:  88        DEY            Schleife und Displ. um 1 vermindern
F3D0:  D0 F7     BNE  $F3C9     Schleifen bis U0;Chr$(31) an Floppy
F3D2:  20 AE F7  JSR  $F7AE     Zeichen des Dateinamens holen
F3D5:  20 D2 FF  JSR  $FFD2     Kernal BSOUT: Ein Zeichen ausgeben
F3D8:  C8        INY            Displacement auf Dateiname erhöhen
F3D9:  C4 9F     CPY  * $9F     Mit Länge d. Dateinamens vergleichen
F3DB:  D0 F5     BNE  $F3D2     Nicht erreicht, nächstes Zeichen
F3DD:  20 CC FF  JSR  $FFCC     Kernal CLRCH: E/A Kanäle rücksetzen
F3E0:  2C 1C 0A  BIT  $0A1C     Prüfe den "Fast Seriell Mode" Zeiger
F3E3:  70 05     BVS  $F3EA     Schnelle Übertragung möglich, Skip
F3E5:  20 8C F4  JSR  $F48C     Logische Datei wieder schließen
F3E8:  38        SEC            Kennzeichen für OK setzen
F3E9:  60        RTS            Rücksprung aus dem Unterprogramm

******************************  LOAD / VERIFY Routinen in Burst Mode

F3EA:  A5 9F     LDA  * $9F     Zwischenspeicher für Dateinamenlänge
F3EC:  85 B7     STA  * $B7     in Z-Page Zeiger für Dateinamenlänge
F3EE:  78        SEI            Alle System Interrupts verhindern
F3EF:  20 45 E5  JSR  $E545     Clock Hi Signal an seriellen Bus
F3F2:  20 C3 E5  JSR  $E5C3     Rückmeldung vom Bus abwarten
F3F5:  2C 0D DC  BIT  $DC0D     Löschen des CIA IRQ-Flags
F3F8:  20 03 F5  JSR  $F503     Clock Lo/Hi Signal invertieren
F3FB:  20 BA F4  JSR  $F4BA     Byte vom Bus holen (Transfer Status)
F3FE:  C9 02     CMP  # $02     Prüfe, ob Transfer Status "File not
F400:  D0 08     BNE  $F40A     found" anzeigt. Nein, dann Skip
F402:  20 8C F4  JSR  $F48C     Clock Hi an Bus und Datei schließen
F405:  68        PLA            Die 2 Byte RTS Rücksprungadresse
F406:  68        PLA            vom Stapel löschen
F407:  4C 85 F6  JMP  $F685     I/O Error #4 (File not found)
F40A:  48        PHA            Gelesenen Transfer Status auf Stack
F40B:  C9 1F     CMP  # $1F     Ist es Kennzeichen f. letzten Block?
```

```
F40D:  D0 0B       BNE  $F41A      Nein, dann Skip Sonderregelung
F40F:  20 03 F5    JSR  $F503      Clock Lo/Hi Signal invertieren
F412:  20 BA F4    JSR  $F4BA      Byte v. Bus holen (Block-Byteanzahl)
F415:  85 A5       STA  * $A5      In Z-Page Bytezahl Schleifenzähler
F417:  4C 21 F4    JMP  $F421      Ladeadresse setzen
F41A:  C9 02       CMP  # $02      Prüfe gelesenen Transfer Status
F41C:  90 03       BCC  $F421      Code $01 zeigt OK an. OK, dann Skip
F41E:  68          PLA             Gesicherten Transfer Status löschen
F41F:  B0 77       BCS  $F498      Sprung zum "Load Error" Exit
F421:  20 33 F5    JSR  $F533      "LOADING"/"VERIFYING" Text ausgeben
F424:  20 03 F5    JSR  $F503      Clock Lo/Hi Signal invertieren
F427:  20 BA F4    JSR  $F4BA      Byte vom Bus holen (Ladeadresse Lo)
F42A:  85 AE       STA  * $AE      In Z-Page Adreßzeiger Lo sichern
F42C:  20 03 F5    JSR  $F503      Clock Lo/Hi Signal invertieren
F42F:  20 BA F4    JSR  $F4BA      Byte vom Bus holen (Ladeadresse Hi)
F432:  85 AF       STA  * $AF      In Z-Page Adreßzeiger Hi sichern
F434:  A6 9E       LDX  * $9E      Lade und prüfe gesicherte Sek.adr.
F436:  D0 08       BNE  $F440      Nicht Null, dann Prg. absolut laden
F438:  A5 C3       LDA  * $C3      LOAD Ladeadresse Lo in Akku holen
F43A:  A6 C4       LDX  * $C4      LOAD Ladeadresse Hi in X-Reg holen
F43C:  85 AE       STA  * $AE      Ladeadresse Lo in Adreßzeiger Lo
F43E:  86 AF       STX  * $AF      Ladeadresse Hi in Adreßzeiger Hi
F440:  A5 AE       LDA  * $AE      Adreßzeiger Lo in Akku holen
F442:  A6 AF       LDX  * $AF      Adreßzeiger Hi in X-Reg holen
F444:  85 AC       STA  * $AC      Akku  als Ladeadreßzeiger Lo setzen
F446:  86 AD       STX  * $AD      X-Reg als Ladeadreßzeiger Hi setzen
F448:  68          PLA             Transfer Status v. Stack zurückholen
F449:  C9 1F       CMP  # $1F      Zeigte Status auf letzten Prg.Block?
F44B:  F0 32       BEQ  $F47F      Ja, dann Skip Standard Blocklänge
F44D:  20 03 F5    JSR  $F503      Clock Lo/Hi Signal invertieren
F450:  A9 FC       LDA  # $FC      Datenbytezähler für ersten Block des
F452:  85 A5       STA  * $A5      zu lesenden Prg.Files auf 252 setzen
F454:  20 3D F6    JSR  $F63D      Shift RUN/STOP Tastaturabfrage
F457:  20 E1 FF    JSR  $FFE1      Kernal STOP: Auf Stop Taste prüfen
F45A:  F0 4A       BEQ  $F4A6      Bei Null Exit durch STOP Taste
F45C:  20 C5 F4    JSR  $F4C5      Block von Disk lesen und verarbeiten
F45F:  B0 51       BCS  $F4B2      Fehler in Speicheradresse, dann RTS
F461:  20 BA F4    JSR  $F4BA      Byte von Bus holen (Transfer-Status)
F464:  C9 02       CMP  # $02      Prüfe gelesenen Transfer Status
F466:  90 06       BCC  $F46E      Code $01 zeigt OK an. OK, dann Skip
F468:  C9 1F       CMP  # $1F      War es Status für letzten Block?
F46A:  F0 0B       BEQ  $F477      Ja, dann letzten Block lesen
F46C:  D0 2A       BNE  $F498      Sprung zum "Load Error" Exit
F46E:  20 03 F5    JSR  $F503      Clock Lo/Hi Signal invertieren
F471:  A9 FE       LDA  # $FE      Datenbytezähler für normal zu
F473:  85 A5       STA  * $A5      lesenden Block auf 254 Bytes setzen
F475:  D0 DD       BNE  $F454      Unbedingter Sprung in Leseroutine
```

```
******************************   Letzten Block im Burst Mode lesen

F477:  20 03 F5   JSR  $F503     Clock Lo/Hi Signal invertieren
F47A:  20 BA F4   JSR  $F4BA     Byte v. Bus holen (Block-Byteanzahl)
F47D:  85 A5      STA  * $A5     In Z-Page Bytezahl Schleifenzähler
F47F:  20 03 F5   JSR  $F503     Clock Lo/Hi Signal invertieren
F482:  20 C5 F4   JSR  $F4C5     Block von Disk lesen und verarbeiten
F485:  B0 2B      BCS  $F4B2     Fehler in Speicheradresse, dann RTS
F487:  A9 40      LDA  # $40     EOF Kennzeichen Code in Akku bringen
F489:  20 57 F7   JSR  $F757     Kernal SETMSG: System Status setzen

******************************   Clock Hi an Bus und Datei schließen

F48C:  20 45 E5   JSR  $E545     Clock Hi Signal an seriellen Bus
F48F:  58         CLI            Alle System Interrupts freigeben
F490:  A5 B8      LDA  * $B8     Logische Dateinummer in Akku holen
F492:  38         SEC            Carry Flag für CLOSE Routine setzen
F493:  20 C3 FF   JSR  $FFC3     Kernal CLOSE: Datei schließen
F496:  18         CLC            Kennzeichen für OK setzen
F497:  60         RTS            Rücksprung aus dem Unterprogramm

******************************   Allgemeiner "Load Error" Exit

F498:  A9 02      LDA  # $02     Fehlercode für Timeout beim Lesen
F49A:  20 57 F7   JSR  $F757     Kernal SETMSG: System Status setzen
F49D:  20 8C F4   JSR  $F48C     Clock Hi an Bus und Datei schließen
F4A0:  68         PLA            Die auf dem Stapel gesicherte RTS
F4A1:  68         PLA            Rücksprungadresse wieder löschen
F4A2:  A9 29      LDA  # $29     Fehler Nr. für BASIC-Fehler: "LOAD"
F4A4:  38         SEC            Setze Kennzeichen f. Fehler gefunden
F4A5:  60         RTS            Rücksprung aus dem Unterprogramm

******************************   Exit bei STOP-Tasten Unterbrechung

F4A6:  20 8C F4   JSR  $F48C     Clock Hi an Bus und Datei schließen
F4A9:  A9 00      LDA  # $00     Z-Page Zeiger für die aktuelle
F4AB:  85 B9      STA  * $B9     Sekundäradresse auf #0 setzen
F4AD:  68         PLA            Die auf dem Stapel gesicherte RTS
F4AE:  68         PLA            Rücksprungadresse wieder löschen
F4AF:  4C B5 F5   JMP  $F5B5     Routine: Exit durch Break Taste

******************************   Exit bei Fehler in Speicheradresse

F4B2:  20 8C F4   JSR  $F48C     Clock Hi an Bus und Datei schließen
F4B5:  68         PLA            Die auf dem Stapel gesicherte RTS
F4B6:  68         PLA            Rücksprungadresse wieder löschen
F4B7:  4C 97 F6   JMP  $F697     Zur Ausgabe des I/O Error's #10
```

```
******************************      Ein Datenbyte im Burst Mode lesen

F4BA:  A9 08      LDA  # $08        Kontrollbit für Bus-Interrupt setzen
F4BC:  2C 0D DC   BIT  $DC0D        Interrupt Steuerregister auslesen
F4BF:  F0 FB      BEQ  $F4BC        und ser. Bus Interrupt abwarten
F4C1:  AD 0C DC   LDA  $DC0C        CIA-Datenpuffer v. ser. Bus auslesen
F4C4:  60         RTS               Rücksprung aus dem Unterprogramm

******************************      Datenblock im Burst Mode lesen

F4C5:  A9 08      LDA  # $08        Kontrollbit für Bus-Interrupt setzen
F4C7:  2C 0D DC   BIT  $DC0D        Interrupt Steuerregister auslesen
F4CA:  F0 FB      BEQ  $F4C7        und ser. Bus Interrupt abwarten
F4CC:  AC 0C DC   LDY  $DC0C        CIA-Datenpuffer v. ser. Bus auslesen
F4CF:  AD 00 DD   LDA  $DD00        Datenport A des CIA 2 auslesen, das
F4D2:  49 10      EOR  # $10        Clock Signal entsprechend inver-
F4D4:  8D 00 DD   STA  $DD00        tieren und in Port A zurückschreiben
F4D7:  98         TYA               Gelesenen Datenpuffer in Akku kop.
F4D8:  A4 93      LDY  * $93        Teste Z-Page LOAD / VERIFY Zeiger
F4DA:  F0 12      BEQ  $F4EE        Bei $00 ist es eine LOAD Routine
F4DC:  85 BD      STA  * $BD        Datenbyte für Verify Oper. speichern
F4DE:  A0 00      LDY  # $00        Displacementzeiger für FETCH Routine
F4E0:  20 C9 F7   JSR  $F7C9        FETCH Routine für LSV Operationen
F4E3:  C5 BD      CMP  * $BD        Vgl. gelesenes Datenbyte m. Speicher
F4E5:  F0 0A      BEQ  $F4F1        Beide gleich, dann OK und weiter
F4E7:  A9 10      LDA  # $10        Nicht gleich, dann Fehler KZ setzen
F4E9:  20 57 F7   JSR  $F757        Kernal STATUS: System Status setzen
F4EC:  D0 03      BNE  $F4F1        Skip STASH Routine (für LOAD)
F4EE:  20 BF F7   JSR  $F7BF        STASH Routine für LSV Operationen
F4F1:  E6 AE      INC  * $AE        Lo-Wert der E-/A- Adresse erhöhen
F4F3:  D0 08      BNE  $F4FD        Kein Überlauf aufgetreten, Skip
F4F5:  E6 AF      INC  * $AF        Hi-Wert der E-/A- Adresse erhöhen
F4F7:  A5 AF      LDA  * $AF        Prüfe, ob der Hi-Wert der E-/A-
F4F9:  C9 FF      CMP  # $FF        Adresse in Systemvektortabelle zeigt
F4FB:  F0 05      BEQ  $F502        Ja, dann ungültig und Skip nach RTS
F4FD:  C6 A5      DEC  * $A5        Datenbyte-Zähler um 1 vermindern
F4FF:  D0 C4      BNE  $F4C5        Schleifen, bis alle Bytes gelesen
F501:  18         CLC               Kennzeichen für "alles OK" setzen
F502:  60         RTS               Rücksprung aus dem Unterprogramm

******************************      Clock Signal am Port A invertieren

F503:  AD 00 DD   LDA  $DD00        Datenport A des CIA 2 auslesen, das
F506:  49 10      EOR  # $10        Clock Signal invertieren und wieder
F508:  8D 00 DD   STA  $DD00        in den Port A zurückschreiben
F50B:  60         RTS               Rücksprung aus dem Unterprogramm
```

```
******************************     Steuersequenz an Floppy in negativer
                                   Reihenfolge. Sendet: U0;Chr$(31)

F50C:  1F 30 55                    <Chr$(31)> <0> <U>

******************************     Steuermeldung "SEARCHING" "FOR"
                                   <Dateiname> ausgeben

F50F:  A5 9D     LDA  * $9D        Zeiger, ob Steuermeldungen erlaubt
F511:  10 1F     BPL  $F532        Nicht erlaubt, dann Rücksprung
F513:  A0 0C     LDY  # $0C        Displacement auf "SEARCHING" Text
F515:  20 22 F7  JSR  $F722        Syst.-/Steuermeldung ausgeben
F518:  A5 B7     LDA  * $B7        Länge des Dateinamens in Akku holen
F51A:  F0 16     BEQ  $F532        Länge gleich 0, dann Rücksprung
F51C:  A0 17     LDY  # $17        Displacement auf "FOR" Text
F51E:  20 22 F7  JSR  $F722        Syst.-/Steuermeldung ausgeben

******************************     Dateinamen ausgeben

F521:  A4 B7     LDY  * $B7        Hole Länge des aktuellen Dateinamens
F523:  F0 0D     BEQ  $F532        Länge = 0, dann Skip
F525:  A0 00     LDY  # $00        Displacement auf Dateinamen init.
F527:  20 AE F7  JSR  $F7AE        1 Byte des Dateinamens holen
F52A:  20 D2 FF  JSR  $FFD2        Kernal BSOUT: Ein Zeichen ausgeben
F52D:  C8        INY               Displ. auf Dateinamenanfang erhöhen
F52E:  C4 B7     CPY  * $B7        Vergleiche mit Länge des Dateinamens
F530:  D0 F5     BNE  $F527        Nicht gleich, dann nächstes Zeichen
F532:  60        RTS               Rücksprung aus dem Unterprogramm

******************************     "LOADING"/"VERIFYING" Text ausgeben

F533:  A0 49     LDY  # $49        Displacement auf "LOADING" Text
F535:  A5 93     LDA  * $93        LOAD-VERIFY KZ aus Z-Page holen
F537:  F0 02     BEQ  $F53B        Wenn LOAD KZ (0), dann ausgeben
F539:  A0 59     LDY  # $59        Displacement auf "VERIFY" Text
F53B:  4C 1E F7  JMP  $F71E        Syst.-/Steuermeldung ausgeben

******************************     Kernal Routine: SAVESP
                                   Abspeichern eines Speicherbereichs

F53E:  86 AE     STX  * $AE        Lo-Adr von "Speichern bis" sichern
F540:  84 AF     STY  * $AF        Hi-Adr von "Speichern bis" sichern
F542:  AA        TAX               Z-Page Adresse "Speichern von" n. X
F543:  B5 00     LDA  * $00,X      Hole Z-Page Adresse "Von" Lo-Wert
F545:  85 C1     STA  * $C1        und in "Speichern von" Lo sichern
F547:  B5 01     LDA  * $01,X      Hole Z-Page Adresse "Von" Hi-Wert
F549:  85 C2     STA  * $C2        und in "Speichern von" Hi sichern
F54B:  6C 32 03  JMP  ($0332)      Vektor zeigt auf SAVESP Rout.($F54E)
```

```
F54E:  A5 BA     LDA  * $BA     Geräteadresse in Akku laden
F550:  C9 01     CMP  # $01     Ist Ausgabeberät die Datasette?
F552:  F0 74     BEQ  $F5C8     Ja, dann in Kassetten Save Routine
F554:  C9 04     CMP  # $04     Geräteadresse kleiner 4?
F556:  B0 09     BCS  $F561     Nein, dann Skip Fehlermeldung
F558:  4C 94 F6  JMP  $F694     I/O Error #9 (Illegal device number)
F55B:  4C 91 F6  JMP  $F691     I/O Error #8 (Missing filename)
F55E:  4C 85 F6  JMP  $F685     I/O Error #4 (File not found)
F561:  A4 B7     LDY  * $B7     Länge des Dateinamens in Y-Reg
F563:  F0 F6     BEQ  $F55B     Länge = 0, dann I/O Error #8 ausgeb.
F565:  A9 61     LDA  # $61     Sekundäradresse auf Print-Write
F567:  85 B9     STA  * $B9     In Z-Page Speicher f.Sekundäradresse
F569:  20 CB F0  JSR  $F0CB     Teste Länge und Sekundäradresse
F56C:  20 BC F5  JSR  $F5BC     Wenn erlaubt, "SAVING" Msg ausgeben
F56F:  A5 BA     LDA  * $BA     Geräteadresse in Akku laden
F571:  20 3E E3  JSR  $E33E     Routine LISTN: LISTN Bef.an ser. Bus
F574:  A5 B9     LDA  * $B9     Sekundäradresse in Akku laden
F576:  20 D2 E4  JSR  $E4D2     Routine SECND: Sekundäradr. f. LISTN
F579:  A0 00     LDY  # $00     Y-Reg als Displ. auf 0 setzen
F57B:  20 51 ED  JSR  $ED51     Startadr.v.C1,C2 nach AD,AC kopieren
F57E:  20 03 E5  JSR  $E503     Routine CIOUT: Byte auf ser. Bus
F581:  A5 AD     LDA  * $AD     Startadresse Hi-Wert speichern
F583:  20 03 E5  JSR  $E503     Routine CIOUT: Byte auf ser. Bus
F586:  20 B7 EE  JSR  $EEB7     Subtr.: Startadresse - Endadresse
F589:  B0 10     BCS  $F59B     Endadresse erreicht, dann Exit
F58B:  20 CC F7  JSR  $F7CC     Startadresse in FETVEC ablegen
F58E:  20 03 E5  JSR  $E503     Routine CIOUT: Byte auf ser. Bus
F591:  20 E1 FF  JSR  $FFE1     Kernal STOP: Auf Stop Taste prüfen
F594:  F0 1F     BEQ  $F5B5     Wenn gedrückt, SAVESP abbrechen
F596:  20 C1 EE  JSR  $EEC1     Startadresse ($AC,$AD) um 1 erhöhen
F599:  D0 EB     BNE  $F586     Überlauf im Hi Byte, dann Exit
F59B:  20 26 E5  JSR  $E526     Routine UNLSN: UNLSN Bef.an ser. Bus
F59E:  24 B9     BIT  * $B9     Teste Bit 7 der Sekundäradresse
F5A0:  30 11     BMI  $F5B3     Ist Bit 7 gesetzt, dann Skip
F5A2:  A5 BA     LDA  * $BA     Geräteadresse in Akku laden
F5A4:  20 3E E3  JSR  $E33E     Routine LISTN: LISTN Bef.an ser. Bus
F5A7:  A5 B9     LDA  * $B9     Sekundäradresse in Akku laden
F5A9:  29 EF     AND  # $EF     Untere Tetrade der SA erhalten und
F5AB:  09 E0     ORA  # $E0     über obere CLOSE an Gerät schicken
F5AD:  20 D2 E4  JSR  $E4D2     Routine SECND: Sekundäradr. f. LISTN
F5B0:  20 26 E5  JSR  $E526     Routine UNLSN: UNLSN Bef.an ser. Bus
F5B3:  18        CLC            Kennzeichen für "OK" setzen
F5B4:  60        RTS            Rücksprung aus dem Unterprogramm

******************************  SAVESP Exit durch Break

F5B5:  20 9E F5  JSR  $F59E     Schreibkanal an Gerät schließen
F5B8:  A9 00     LDA  # $00     Akku mit $00 als KZ laden
```

```
F5BA:  38          SEC             Carry für Break / Fehler KZ setzen
F5BB:  60          RTS             Rücksprung aus dem Unterprogramm

******************************     Prüfe, ob SAVING Steuermeldung
                                   ausgegeben werden kann

F5BC:  A5 9D       LDA  * $9D      Prüfe, ob Steuermeldungen erlaubt
F5BE:  10 37       BPL  $F5F7      Nicht erlaubt, dann Rücksprung
F5C0:  A0 51       LDY  # $51      Displ. auf "SAVING" Meldung in Y-Reg
F5C2:  20 22 F7    JSR  $F722      "SAVING" Meldung ausgeben
F5C5:  4C 21 F5    JMP  $F521      und Dateiname ausgeben: RTS

******************************     SAVE Routine für Datasette

F5C8:  20 80 E9    JSR  $E980      Kassettenpufferzeiger in X+Y Reg
F5CB:  90 8B       BCC  $F558      Page 0,1 nicht erlaubt: I/O Error #9
F5CD:  20 E9 E9    JSR  $E9E9      Warten auf "Record & Play" Tasten
F5D0:  B0 25       BCS  $F5F7      STOP, dann Abbruch
F5D2:  20 BC F5    JSR  $F5BC      Wenn erlaubt, "SAVING" Msg ausgeben
F5D5:  A2 03       LDX  # $03      Header Typ 3= Maschinenprg (absolut)
F5D7:  A5 B9       LDA  * $B9      Sekundäradresse in Akku laden
F5D9:  29 01       AND  # $01      Teste, ob Bit 0 gesetzt
F5DB:  D0 02       BNE  $F5DF      Ja, dann Maschinenprogramm
F5DD:  A2 01       LDX  # $01      Header Typ 1= BASIC-Prg
F5DF:  8A          TXA             Header Typ in Akku kopieren
F5E0:  20 19 E9    JSR  $E919      und Header auf Band schreiben
F5E3:  B0 12       BCS  $F5F7      Exit, wenn Stop Taste gedrückt
F5E5:  20 18 EA    JSR  $EA18      Programm auf Kassette speichern
F5E8:  B0 0D       BCS  $F5F7      Exit, wenn Stop Taste gedrückt
F5EA:  A5 B9       LDA  * $B9      Sekundäradresse in Akku laden
F5EC:  29 02       AND  # $02      Prüfe, ob Bit 1 gesetzt ist
F5EE:  F0 06       BEQ  $F5F6      nicht gesetzt, dann "OK" Exit
F5F0:  A9 05       LDA  # $05      Code für EOT Kontrollbyte in Akku
F5F2:  20 19 E9    JSR  $E919      und Block auf Band schreiben
F5F5:  24          .Byte  $24      Skip nach $F5F7
F5F6:  18          CLC             Kennzeichen für "OK" setzen
F5F7:  60          RTS             Rücksprung aus dem Unterprogramm

******************************     Kernal Routine: UDTIM
                                   Update der internen 24h Uhr

F5F8:  E6 A2       INC  * $A2      Lo Byte der 24 Stunden System Uhr +1
F5FA:  D0 06       BNE  $F602      Kein Überlauf,Skip Überlaufkorrektur
F5FC:  E6 A1       INC  * $A1      Mi Byte der 24 Stunden System Uhr +1
F5FE:  D0 02       BNE  $F602      Kein Überlauf,Skip Überlaufkorrektur
F600:  E6 A0       INC  * $A0      Hi Byte der 24 Stunden System Uhr +1
F602:  38          SEC             Carry für Subtraktion setzen
F603:  A5 A2       LDA  * $A2      Durch Subtraktion der entsprechenden
```

```
F605:  E9 01      SBC  # $01     Werte wird hier geprüft, ob die
F607:  A5 A1      LDA  * $A1     interne 24 Stunden System Uhr in
F609:  E9 1A      SBC  # $1A     den Bytes $A0-$A1-$A2 auf die
F60B:  A5 A0      LDA  * $A0     Uhrzeit 24.00.00 gesetzt wurde, da
F60D:  E9 4F      SBC  # $4F     in diesem Falle die 3 Bytes wieder
F60F:  90 08      BCC  $F619     neu initialisiert werden müssen.
F611:  A2 00      LDX  # $00     24 Stunden System Uhr auf 00.00.00
F613:  86 A0      STX  * $A0     Zero-Page Byte für System Uhr High
F615:  86 A1      STX  * $A1     Zero-Page Byte für System Uhr Middle
F617:  86 A2      STX  * $A2     Zero-Page Byte für System Uhr Low
F619:  AD 1D 0A   LDA  $0A1D     Prüfe Zwischenspeicher f. 24h Uhr Lo
F61C:  D0 0B      BNE  $F629     Nicht Null, dann nur Lo Wert -1
F61E:  AD 1E 0A   LDA  $0A1E     Prüfe Zwischenspeicher f. 24h Uhr Mi
F621:  D0 03      BNE  $F626     Nicht Null, dann nur Lo und Mi -1
F623:  CE 1F 0A   DEC  $0A1F     Zwischenspeicher für 24h Uhr Hi -1
F626:  CE 1E 0A   DEC  $0A1E     Zwischenspeicher für 24h Uhr Mi -1
F629:  CE 1D 0A   DEC  $0A1D     Zwischenspeicher für 24h Uhr Lo -1
F62C:  2C 03 0A   BIT  $0A03     Teste PAL / NTSC Zeiger
F62F:  10 0C      BPL  $F63D     Bei "plus" ist es ein NTSC System
F631:  CE 36 0A   DEC  $0A36     Rasterzeilen Zeilenzeiger -1
F634:  10 07      BPL  $F63D     Noch nicht Null, dann Skip Initial.
F636:  A9 05      LDA  # $05     System-Zeiger f. Rasterzeile, an der
F638:  8D 36 0A   STA  $0A36     Interrupt ausgelöst wird mit 5 init.
F63B:  D0 BB      BNE  $F5F8     Unbed. Sprung in neuen UDTIM Aufruf

******************************   Tastaturabfrage Reihenauswahl auf
                                 RUN/STOP und SHIFT Tasten

F63D:  AD 01 DC   LDA  $DC01     Port B für Tastaturmatrixspalten
F640:  CD 01 DC   CMP  $DC01     auslesen und warten, bis Tastatur-
F643:  D0 F8      BNE  $F63D     matrix entprellt ist
F645:  AA         TAX            Gelesenen Tastencode nach X-Reg und
F646:  30 13      BMI  $F65B     Skip, wenn RUN/STOP gedrückt ist
F648:  A2 BD      LDX  # $BD     Bitmuster für SHIFT Reihenauswahl
F64A:  8E 00 DC   STX  $DC00     In Port A für Matrixzeilenauswahl
F64D:  AE 01 DC   LDX  $DC01     Port B für Tastaturmatrixspalten
F650:  EC 01 DC   CPX  $DC01     auslesen und warten, bis Tastatur-
F653:  D0 F8      BNE  $F64D     matrix entprellt ist
F655:  8D 00 DC   STA  $DC00     In Port A für Matrixzeilenauswahl
F658:  E8         INX            Gelesenen Wert um 1 erhöhen
F659:  D0 02      BNE  $F65D     Keine der beiden Shift-Tasten, Skip
F65B:  85 91      STA  * $91     Z-Page Stop / RVS Zeiger neu setzen
F65D:  60         RTS            Rücksprung aus dem Unterprogramm

******************************   Kernal Routine: RDTIM
                                 24 Stunden System-Uhr lesen

F65E:  78         SEI            Alle System Interrupts verhindern
```

```
F65F:  A5 A2     LDA  * $A2      Z-Page Byte für System-Uhr Low
F661:  A6 A1     LDX  * $A1      Z-Page Byte für System-Uhr Middle
F663:  A4 A0     LDY  * $A0      Z-Page Byte für System-Uhr High

******************************   Kernal Routine: SETTIM
                                 24 Stunden System Uhr-setzen

F665:  78        SEI             Alle System Interrupts verhindern
F666:  85 A2     STA  * $A2      Z-Page Byte für System-Uhr Low
F668:  86 A1     STX  * $A1      Z-Page Byte für System-Uhr Middle
F66A:  84 A0     STY  * $A0      Z-Page Byte für System-Uhr High
F66C:  58        CLI             Alle System Interrupts freigeben
F66D:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   Kernal Routine: STOP
                                 Auf gedrückte STOP Taste prüfen

F66E:  A5 91     LDA  * $91      Z-Page Speicher f. STOP Flag holen
F670:  C9 7F     CMP  # $7F      Wurde STOP Taste gedrückt?
F672:  D0 07     BNE  $F67B      Nein, Rücksprung mit Equal Flag 0
F674:  08        PHP             Status des Equal Flag retten
F675:  20 CC FF  JSR  $FFCC      Kernal CLRCH: E/A Kanäle rücksetzen
F678:  85 D0     STA  * $D0      Z-Page Tastaturpuffer Zeiger löschen
F67A:  28        PLP             Status des Equal Flag zurückholen
F67B:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   I/O Error Message ausgeben

F67C:  A9 01     LDA  # $01      I/O Error #1 (Too many files)
F67E:  2C        .Byte  $2C      Skip nach $F681
F67F:  A9 02     LDA  # $02      I/O Error #2 (File open)
F681:  2C        .Byte  $2C      Skip nach $F684
F682:  A9 03     LDA  # $03      I/O Error #3 (File not open)
F684:  2C        .Byte  $2C      Skip nach $F687
F685:  A9 04     LDA  # $04      I/O Error #4 (File not found)
F687:  2C        .Byte  $2C      Skip nach $F68A
F688:  A9 05     LDA  # $05      I/O Error #5 (Device not present)
F68A:  2C        .Byte  $2C      Skip nach $F68D
F68B:  A9 06     LDA  # $06      I/O Error #6 (Not input file)
F68D:  2C        .Byte  $2C      Skip nach $F690
F68E:  A9 07     LDA  # $07      I/O Error #7 (Not output file)
F690:  2C        .Byte  $2C      Skip nach $F693
F691:  A9 08     LDA  # $08      I/O Error #8 (Missing filename)
F693:  2C        .Byte  $2C      Skip nach $F696
F694:  A9 09     LDA  # $09      I/O Error #9 (Illegal device number)
F696:  2C        .Byte  $2C      Skip nach $F699
F697:  A9 10     LDA  # $10      I/O Error #0 (Break by STOP Key)
F699:  48        PHA             I/O Error-Code auf Stack sichern
```

```
F69A:  20 CC FF    JSR  $FFCC        Kernal CLRCH: E/A Kanäle rücksetzen
F69D:  A0 00       LDY  # $00        Displacement auf I/O Error Meldung
F69F:  24 9D       BIT  * $9D        Prüfe, ob System-Meldungen erlaubt
F6A1:  50 0A       BVC  $F6AD        Nicht erlaubt, dann Exit
F6A3:  20 22 F7    JSR  $F722        Routine:Syst/Steuermeldungen ausgeb.
F6A6:  68          PLA               Error Code Nummer in Akku holen
F6A7:  48          PHA               und wieder auf Stack sichern
F6A8:  09 30       ORA  # $30        ASCII Wert des Error-Codes erzeugen
F6AA:  20 D2 FF    JSR  $FFD2        Kernal BSOUT: Ein Zeichen ausgeben
F6AD:  68          PLA               Error-Code vom Stack löschen
F6AE:  38          SEC               Carry Flag als Kennzeichen setzen
F6AF:  60          RTS               Rücksprung aus dem Unterprogramm

*******************************      Tab. der System- und Steuermeldungen
                                     In Klammern der Offset zum Anfang

F6B0:  0D 49 2F 4F 20 45 52 52       <Cr>  I/O ERROR #                 ($00)
F6B8:  4F 52 20 A3
FCBC:  0D 53 45 41 52 43 48 49       <Cr>  SEARCHING                   ($0C)
F6C4:  4E 47 A0
F6C7:  46 4F 52 A0                   FOR                               ($17)
F6CB:  0D 50 52 45 53 53 20 50       <Cr>  PRESS PLAY ON TAPE          ($1B)
F6D3:  4C 41 59 20 4F 4E 20 54
F6DB:  41 50 C5
F6DE:  50 52 45 53 53 20 52 45       PRESS RECORD & PLAY ON TAPE       ($2E)
F6E6:  43 4F 52 44 20 26 20 50
F6EE:  4C 41 59 20 4F 4E 20 54
F6F6:  41 50 C5
F6F9:  0D 4C 4F 41 44 49 4E C7       <Cr>  LOADING                     ($49)
F701:  0D 53 41 56 49 4E 47 A0       <Cr>  SAVING                      ($51)
F709:  0D 56 45 52 49 46 59 49       <Cr>  VERIFYING                   ($59)
F711:  4E C7
F713:  0D 46 4F 55 4E 44 A0          <Cr>  FOUND                       ($63)
F71A:  0D 4F 4B 8D                   <Cr>  OK  <Cr>                    ($6A)

*******************************      Syst-/Steuermeldung ausgeben

F71E:  24 9D       BIT  * $9D        Prüfe, ob Ausgabe erlaubt
F720:  10 0D       BPL  $F72F        Nein, dann Exit
F722:  B9 B0 F6    LDA  $F6B0,Y      Byte aus Meldungstabelle lesen
F725:  08          PHP               und auf dem Stapel sichern
F726:  29 7F       AND  # $7F        Bit 7 ausblenden, keine RVS Zeichen
F728:  20 D2 FF    JSR  $FFD2        Kernal BSOUT: Ein Zeichen ausgeben
F72B:  C8          INY               Displ. auf nächstes Zeichen setzen
F72C:  28          PLP               Zeichen vom Stapel zurückholen
F72D:  10 F3       BPL  $F722        Bit 7 gesetzt ist Ende Kennzeichen
F72F:  18          CLC               Carry als 'ausgegeben' KZ löschen
```

```
F730:  60        RTS           Rücksprung aus dem Unterprogramm

******************************  Kernal Routine: SETNAM
                               Parameter für Dateinamen setzen

F731:  85 B7     STA  * $B7    Z-Page Byte für Länge d. Dateinamens
F733:  86 BB     STX  * $BB    Z-Page Byte für Dateinamen Adr-Lo
F735:  84 BC     STY  * $BC    Z-Page Byte für Dateinamen Adr-Hi
F737:  60        RTS           Rücksprung aus dem Unterprogramm

******************************  Kernal Routine: SETLFS
                               Setzen der logischen Dateiparameter

F738:  85 B8     STA  * $B8    Z-Page Byte für logische Dateinummer
F73A:  86 BA     STX  * $BA    Z-Page Byte für Geräteadresse
F73C:  84 B9     STY  * $B9    Z-Page Byte für Sekundäradresse
F73E:  60        RTS           Rücksprung aus dem Unterprogramm

******************************  Kernal Routine: SETBNK

F73F:  85 C6     STA  * $C6    Bank Nr. f. aktuellen LSV Aufruf
F741:  86 C7     STX  * $C7    Bank Nr. f. aktuellen Dateinamen
F743:  60        RTS           Rücksprung aus dem Unterprogramm

******************************  Kernal Routine: READST
                               System Statuswort lesen

F744:  A5 BA     LDA  * $BA    Geräteadresse in Akku laden
F746:  C9 02     CMP  # $02    Wird die RS-232 angesprochen
F748:  D0 0B     BNE  $F755    Nein, dann normalen Status holen
F74A:  AD 14 0A  LDA  $0A14    RS-232 Status holen
F74D:  48        PHA           und auf dem Stapel sichern
F74E:  A9 00     LDA  # $00    Akku mit $00 laden und in RS-232
F750:  8D 14 0A  STA  $0A14    Status als alles OK KZ bringen
F753:  68        PLA           RS-232 Status v. Stack zurückholen
F754:  60        RTS           Rücksprung aus dem Unterprogramm

******************************  Status auf System Status abstimmen

F755:  A5 90     LDA  * $90    System Status in Akku laden
F757:  05 90     ORA  * $90    Akku mit System Status verknüpfen
F759:  85 90     STA  * $90    In Z-Page f. Status bringen
F75B:  60        RTS           Rücksprung aus dem Unterprogramm

******************************  Kernal Routine: SETMSG
                               System-/Steuermeldungen ermöglichen

F75C:  85 9D     STA  * $9D    Z-Page Byte für Syst/Steuermeldungen
```

```
F75E:  60          RTS           Rücksprung aus dem Unterprogramm

******************************   Kernal Routine: SETTMO
                                 Um Timeout in IEEE Routinen zu
                                 ermöglichen, Bit 7 im Akku auf 1

F75F:  8D 0E 0A    STA  $0A0E    Akku Inhalt in IEEE Timeout Flag
F762:  60          RTS           Rücksprung aus dem Unterprogramm

******************************   Kernal Routine: MEMTOP
                                 Setzen d. oberen Speicherendezeigers

F763:  90 06       BCC  $F76B    Carry 0 = setzen / Carry 1 = lesen
F765:  AE 07 0A    LDX  $0A07    Lo-Adr des RAM Endes in Syst. Bank
F768:  AC 08 0A    LDY  $0A08    Hi-Adr des RAM Endes in Syst. Bank
F76B:  8E 07 0A    STX  $0A07    Lo-Adr des RAM Endes in Syst. Bank
F76E:  8C 08 0A    STY  $0A08    Hi-Adr des RAM Endes in Syst. Bank
F771:  60          RTS           Rücksprung aus dem Unterprogramm

******************************   Kernal Routine: MEMBOT
                                 Unteren Speicherendezeiger setzen

F772:  90 06       BCC  $F77A    Carry 0 = setzen / Carry 1 = lesen
F774:  AE 05 0A    LDX  $0A05    Lo-Adr des RAM Anfangs in Syst. Bank
F777:  AC 06 0A    LDY  $0A06    Hi-Adr des RAM Anfangs in Syst. Bank
F77A:  8E 05 0A    STX  $0A05    Lo-Adr des RAM Anfangs in Syst. Bank
F77D:  8C 06 0A    STY  $0A06    Hi-Adr des RAM Anfangs in Syst. Bank
F780:  60          RTS           Rücksprung aus dem Unterprogramm

******************************   Kernal Routine: IOBASE

F781:  A2 00       LDX  # $00    Adresse Lo d. I/O Bereichs übergeben
F783:  A0 D0       LDY  # $D0    Adresse Hi d. I/O Bereichs übergeben
F785:  60          RTS           Rücksprung aus dem Unterprogramm

******************************   Kernal Routine: LKUPSA
                                 Suche in SA-Tabelle nach SA

F786:  98          TYA           Die zu suchende SA in Akku bringen
F787:  A6 98       LDX  * $98    Zahl der offenen Dateien holen
F789:  CA          DEX           Um 1 vermindern,da als Index benutzt
F78A:  30 0F       BMI  $F79B    Alle Vergleiche negativ, dann Exit
F78C:  DD 76 03    CMP  $0376,X  Mit Byte aus SA-Tabelle vergleichen
F78F:  D0 F8       BNE  $F789    Nicht gefunden, nächster Vergleich
F791:  20 12 F2    JSR  $F212    LFN,GA,SA entsp. X-Reg a. Tab holen
F794:  AA          TAX           Gefundene GA in X-Reg kopieren
F795:  A5 B8       LDA  * $B8    Logische Dateinummer in Akku holen
F797:  A4 B9       LDY  * $B9    Sekundäradresse in Y-Reg holen
```

```
F799:  18          CLC              Carry gelöscht = KZ für gefunden
F79A:  60          RTS              Rücksprung aus dem Unterprogramm

******************************      Exit aus LKUPSA bei "nicht gefunden"

F79B:  38          SEC              Carry gesetzt = KZ f. nicht gefunden
F79C:  60          RTS              Rücksprung aus dem Unterprogramm

******************************      Kernal Routine: LKUPLA
                                    Suche in LFN-Tabelle nach LFN

F79D:  AA          TAX              Zu suchenden LFN Wert in X sichern
F79E:  20 02 F2    JSR  $F202       Status OK setzen, LFN Tabelle suchen
F7A1:  F0 EE       BEQ  $F791       Gefunden, Update der Z-Page, Exit
F7A3:  D0 F6       BNE  $F79B       Nicht gefunden, Exit mit Fehler KZ

******************************      Kernal Routine: DMA-CALL

F7A5:  BD F0 F7    LDA  $F7F0,X     X indiz. Wert aus Config.Tab holen
F7A8:  29 FE       AND  # $FE       Bit 0 ausblenden (I/O v. D000-DFFF)
F7AA:  AA          TAX              Config Wert nach X-Reg kopieren
F7AB:  4C F0 03    JMP  $03F0       Sprung zur Lo-Bank DMA Routine

******************************      FETCH für Zeichen aus Dateinamen

F7AE:  8E 35 0A    STX  $0A35       Inhalt des X-Reg sichern
F7B1:  A6 C7       LDX  * $C7       Bank Nr.f.akt.Dateinamen bei (BB,BC)
F7B3:  A9 BB       LDA  # $BB       $BB für FETVEC in Akku bringen
F7B5:  20 D0 F7    JSR  $F7D0       Rout. INDFET: LDA(fetvec),Y bel.Bank
F7B8:  AE 35 0A    LDX  $0A35       Alten Inhalt X-Reg zurückholen
F7BB:  60          RTS              Rücksprung aus dem Unterprogramm

******************************      STASH Routine für LSV Operationen

F7BC:  A2 AC       LDX  # $AC       Pointer auf LSV E/A Adresse 1 (Lo)
F7BE:  2C          .Byte  $2C       Skip nach $F7C1
F7BF:  A2 AE       LDX  # $AE       Pointer auf LSV E/A Adresse 2 (Lo)
F7C1:  8E B9 02    STX  $02B9       Inhalt d. X-Reg in STAVEC bringen
F7C4:  A6 C6       LDX  * $C6       Bank Nr. d. aktuellen LSV Aufrufs
F7C6:  4C DA F7    JMP  $F7DA       Rout. INDSTA: STA(stavec),Y bel.Bank

******************************      FETCH Routine für LSV Operationen

F7C9:  A9 AE       LDA  # $AE       Pointer auf LSV E/A Adresse 1 (Lo)
F7CB:  2C          .Byte  $2C       Skip nach $F7CE
F7CC:  A9 AC       LDA  # $AC       Pointer auf LSV E/A Adresse 2 (Lo)
F7CE:  A6 C6       LDX  * $C6       Bank Nr. d. aktuellen LSV Aufrufs
```

```
******************************      Vorbereitung der FETCH Routine

F7D0:   8D AA 02   STA  $02AA       Akku Inhalt in FETVEC ablegen
F7D3:   BD F0 F7   LDA  $F7F0,X     Den mit X bestimmten Configuration
F7D6:   AA         TAX              Wert a. Tab. laden und nach X-Reg
F7D7:   4C A2 02   JMP  $02A2       FETCH Rout.: LDA von beliebiger Bank

******************************      Vorbereitung der STASH Routine

F7DA:   48         PHA              Akku Inhalt für STA Befehl sichern
F7DB:   BD F0 F7   LDA  $F7F0,X     Den mit X bestimmten Configuration
F7DE:   AA         TAX              Wert a. Tab. laden und nach X-Reg
F7DF:   68         PLA              Akku Inhalt für STA Befehl laden
F7E0:   4C AF 02   JMP  $02AF       STASH Routine: STA in beliebige Bank

******************************      Vorbereitung der CMPFAR Routine

F7E3:   48         PHA              Akku Inhalt für Vergleich sichern
F7E4:   BD F0 F7   LDA  $F7F0,X     Den mit X bestimmten Configuration
F7E7:   AA         TAX              Wert a. Tab. holen und nach X-Reg
F7E8:   68         PLA              Akku Inhalt für Vergleich holen
F7E9:   4C BE 02   JMP  $02BE       CMPARE Routine: CMP mit belieb. Bank

******************************      Kernal Routine: GETCFG
                                    Mit X definierten Config. Wert laden

F7EC:   BD F0 F7   LDA  $F7F0,X     Mit X definierten Config. Wert laden
F7EF:   60         RTS              Rücksprung aus dem Unterprogramm

******************************      Configuration Tabelle für
                                    alle 'FAR' Operationen

F7F0:   3F    (% 0011 1111)         Bit 0:    0  = I/O Area $D000 - $DFFF
F7F1:   7F    (% 0111 1111)               1  = RAM/ROM Bereich
F7F2:   BF    (% 1011 1111)         Bit 1:    0  = ROM in $4000 - $7FFF
F7F3:   FF    (% 1111 1111)               1  = RAM in $4000 - $7FFF
F7F4:   16    (% 0001 0110)         Bit 3,2: 00 = System ROM $8000-$BFFF
F7F5:   56    (% 0101 0110)              01 = Intern. Funktion ROM
F7F6:   96    (% 1001 0110)              10 = Extern. Funktion ROM
F7F7:   D6    (% 1101 0110)              11 = RAM Bereich
F7F8:   2A    (% 0010 1010)         Bit 5,4: 00 = System ROM $C000-$FFFF
F7F9:   6A    (% 0110 1010)              01 = Intern. Funktion ROM
F7FA:   AA    (% 1010 1010)              10 = Extern. Funktion ROM
F7FB:   EA    (% 1110 1010)              11 = RAM Bereich
F7FC:   06    (% 0000 0110)         Bit 7,6: 00 = RAM Bank 0
F7FD:   0A    (% 0000 1010)              01 = RAM Bank 1
F7FE:   01    (% 0000 0001)              10 = RAM Bank 2 (Bank 0)
F7FF:   00    (% 0000 0000)              11 = RAM Bank 3 (Bank 1)
```

```
******************************    ROM Kopie der FETCH Routine ($02A2)

F800:  AD 00 FF   LDA  $FF00      Aktuellen Config. Wert in A retten
F803:  8E 00 FF   STX  $FF00      Neuen Config. Wert über X setzen
F806:  AA         TAX             Alten Wert nach X transferieren
F807:  B1 FF      LDA  ($FF),Y    !!! LDA (Fetvec) ,Y
F809:  8E 00 FF   STX  $FF00      Alte Config. wiederherstellen
F80C:  60         RTS             Rücksprung aus dem Unterprogramm

******************************    ROM Kopie der STASH Routine ($02AF)

F80D:  48         PHA             Akku Inhalt für STA retten
F80E:  AD 00 FF   LDA  $FF00      Aktuellen Config. Wert in A retten
F811:  8E 00 FF   STX  $FF00      Neuen Config. Wert über X setzen
F814:  AA         TAX             Alten Wert nach X transferieren
F815:  68         PLA             Den STA Wert wieder zurückholen
F816:  91 FF      STA  ($FF),Y    !!! STA (Stavec) ,Y
F818:  8E 00 FF   STX  $FF00      Alte Config. wiederherstellen
F81B:  60         RTS             Rücksprung aus dem Unterprogramm

******************************    ROM Kopie der CMPARE Routine ($02BE)

F81C:  48         PHA             Vergleichswert für CMP retten
F81D:  AD 00 FF   LDA  $FF00      Aktuellen Config. Wert in A retten
F820:  8E 00 FF   STX  $FF00      Neuen Config. Wert über X setzen
F823:  AA         TAX             Alten Wert nach X transferieren
F824:  68         PLA             CMP Vergleichswert zurückholen
F825:  D1 FF      CMP  ($FF),Y    !!! CMP (Cmpvec) ,Y
F827:  8E 00 FF   STX  $FF00      Alte Config. wiederherstellen
F82A:  60         RTS             Rücksprung aus dem Unterprogramm

******************************    ROM Kopie der JSRFAR Routine ($02CD)

F82B:  20 E3 02   JSR  $02E3      JMPFAR Rout.: JMP in beliebige Bank
F82E:  85 06      STA  * $06      Akku in Z-Page Akku Speicher retten
F830:  86 07      STX  * $07      X-Reg in Z-P. X-Reg Speicher retten
F832:  84 08      STY  * $08      Y-Reg in Z-P. Y-Reg Speicher retten
F834:  08         PHP             Prozessor Status auf Stack retten
F835:  68         PLA             Status in Akku zurückholen
F836:  85 05      STA  * $05      u. in Z-Page Status Speicher retten
F838:  BA         TSX             Stack Pointer Zeiger über X-Reg
F839:  86 09      STX  * $09      in Z-P Stack Pointer Speicher retten
F83B:  A9 00      LDA  # $00      Configuration Register mit $00 laden
F83D:  8D 00 FF   STA  $FF00      und alle System ROMs einschalten
F840:  60         RTS             Rücksprung aus dem Unterprogramm
```

```
******************************      ROM Kopie der JMPFAR Routine ($02E3)

F841:  A2 00     LDX  # $00         In dieser Schleife werden die in
F843:  B5 03     LDA  * $03,X       der Zero-Page (Bytes $03-$04-$05)
F845:  48        PHA                abgelegten Werte für den Programm-
F846:  E8        INX                Zähler und den Prozessor Status auf
F847:  E0 03     CPX  # $03         den Stack gebracht, da diese für den
F849:  90 F8     BCC  $F843         RTI am Ende d. Rout. benötigt werden
F84B:  A6 02     LDX  * $02         Bank-Zeiger für Config.-Displ. laden
F84D:  20 6B FF  JSR  $FF6B         Kernal GETCFG: Config. Wert aus Tab.
F850:  8D 00 FF  STA  $FF00         Config. Register entsprechend setzen
F853:  A5 06     LDA  * $06         Zero-Page Akku Speicher holen
F855:  A6 07     LDX  * $07         Zero-Page X-Reg Speicher holen
F857:  A4 08     LDY  * $08         Zero-Page Y-Reg Speicher holen
F859:  40        RTI                Sprung an d. Program-Counter Adresse

******************************      Kopie der Routine in ($03F0)

F85A:  AE 00 FF  LDX  $FF00         Hole Configuration Register in X-Reg
F85D:  8C 01 DF  STY  $DF01         DMA-Controller Steuerregister setzen
F860:  8D 00 FF  STA  $FF00         Config. Reg. mit Akku Inhalt laden
F863:  8E 00 FF  STX  $FF00         Config. Reg. mit X-Reg Inhalt laden
F866:  60        RTS                Rücksprung aus dem Unterprogramm

******************************      Kernal Routine: PHOENIX
                                    Alle Kalt-Start Routinen

F867:  78        SEI                Alle System Interrupts verhindern
F868:  A2 03     LDX  # $03         Bank und Displacementzeiger für
F86A:  8E C0 0A  STX  $0AC0         externe Karten auf #3 initialisieren
F86D:  AE C0 0A  LDX  $0AC0         Displacementzeiger in X-Reg holen
F870:  BD C1 0A  LDA  $0AC1,X       Prüfe ID-Tabelle für Modulplätze
F873:  F0 11     BEQ  $F886         Tab.-Eintrag = 0: Nicht "Logged in"
F875:  A0 00     LDY  # $00         Einsprungadresse Lo auf $00 setzen
F877:  BD BC E2  LDA  $E2BC,X       Hole Einsprungadresse Hi aus Tabelle
F87A:  85 03     STA  * $03         Sichere Einsprungadresse Hi im PC-Hi
F87C:  84 04     STY  * $04         Sichere Einsprungadresse Lo im PC-Lo
F87E:  BD C0 E2  LDA  $E2C0,X       Hole Bank-Wert aus Bank-Tabelle und
F881:  85 02     STA  * $02         sichere ihn im Z-Page Bank-Speicher
F883:  20 CD 02  JSR  $02CD         JSRFAR Rout.: JSR beliebig.Bank +RTS
F886:  CE C0 0A  DEC  $0AC0         Displacementzeiger um 1 vermindern
F889:  10 E2     BPL  $F86D         Alle 4 Modulbereiche abprüfen
F88B:  58        CLI                Alle System Interrupts freigeben
F88C:  A2 08     LDX  # $08         Geräteadr. f. BOOT-Load v. Floppy 8
F88E:  A9 30     LDA  # $30         Akku mit Zeichen <0> laden
F890:  85 BF     STA  * $BF         Z-Page Byte für seriellen Buffer
```

```
F892:  86 BA     STX  * $BA     Geräteadresse für Floppy 8 setzen
F894:  8A        TXA            Kopiere Geräteadresse (8) in Akku
F895:  20 3D F2  JSR  $F23D     Standard E-/A- Geräte setzen
F898:  A2 00     LDX  # $00     Längenzähler für BOOT-Load Datei-
F89A:  86 9F     STX  * $9F     namen mit #0 initialisieren
F89C:  86 C2     STX  * $C2     Sektor Nr. f. BOOT-Load setzen ($00)
F89E:  E8        INX            Initialisierungszähler um 1 erhöhen
F89F:  86 C1     STX  * $C1     Track Nr. für BOOT-Load setzen ($01)
F8A1:  C8        INY            Y-Schleifenregister um 1 erhöhen
F8A2:  D0 FD     BNE  $F8A1     256 mal schleifen, bis Reg. Null ist
F8A4:  E8        INX            X-Schleifenregister um 1 erhöhen
F8A5:  D0 FA     BNE  $F8A1     256 mal schleifen, bis Reg. Null ist
F8A7:  A2 0C     LDX  # $0C     Displacementzeiger für DOS-Puffer
F8A9:  BD 08 FA  LDA  $FA08,X   Zeichen des DOS BOOT-Befehls holen
F8AC:  9D 00 01  STA  $0100,X   und in d. DOS-String Puffer kopieren
F8AF:  CA        DEX            Displacement Zeiger um 1 vermindern
F8B0:  10 F7     BPL  $F8A9     Schleifen, bis 13 Zeichen übertragen
F8B2:  A5 BF     LDA  * $BF     Laufwerksnummer aus Z-Page Speicher
F8B4:  8D 06 01  STA  $0106     holen und im DOS-Puffer ablegen
F8B7:  A9 00     LDA  # $00     Bank Nr. f. aktuellen LSV Aufruf
F8B9:  A2 0F     LDX  # $0F     Bank Nr. f. aktuellen Dateinamen
F8BB:  20 3F F7  JSR  $F73F     Routine SETBNK: Bank für LSV+Datnam.
F8BE:  A9 01     LDA  # $01     Länge des Dateinamens auf 1 setzen
F8C0:  A2 15     LDX  # $15     Adr-Lo des Dateinamens  (=FA15)
F8C2:  A0 FA     LDY  # $FA     Adr-Hi des Dateinamens  ("I")
F8C4:  20 31 F7  JSR  $F731     Routine SETNAM: Setze Dateinamen
F8C7:  A9 00     LDA  # $00     Logische Dateinummer in Akku (0)
F8C9:  A0 0F     LDY  # $0F     Sekundäradresse in Y-Reg
F8CB:  A6 BA     LDX  * $BA     Geräteadresse in X-Reg laden
F8CD:  20 38 F7  JSR  $F738     Routine SETLFS: Setze Dateiparameter
F8D0:  20 C0 FF  JSR  $FFC0     Kernal OPEN: Datei öffnen 0,8,15,"I"
F8D3:  B0 16     BCS  $F8EB     Fehler aufgetreten,BOOT-Load beenden
F8D5:  A9 01     LDA  # $01     Länge des Dateinamens auf 1 setzen
F8D7:  A2 16     LDX  # $16     Adr-Lo des Dateinamens  (=FA16)
F8D9:  A0 FA     LDY  # $FA     Adr-Hi des Dateinamens  ("#")
F8DB:  20 31 F7  JSR  $F731     Routine SETNAM: Setze Dateinamen
F8DE:  A9 0D     LDA  # $0D     Logische Dateinummer in Akku (13)
F8E0:  A8        TAY            Und als Sekundäradresse setzen (13)
F8E1:  A6 BA     LDX  * $BA     Geräteadresse in X-Reg laden
F8E3:  20 38 F7  JSR  $F738     Routine SETLFS: Setze Dateiparameter
F8E6:  20 C0 FF  JSR  $FFC0     Kernal OPEN:Datei öffnen 13,8,13,"#"
F8E9:  90 03     BCC  $F8EE     Alles klar, dann weiter im BOOT-Load
F8EB:  4C 8B F9  JMP  $F98B     Floppy initialisieren, dann RTS
F8EE:  A9 00     LDA  # $00     Den 2-Byte Z-Page Zeiger ($AC-$AD)
F8F0:  A0 0B     LDY  # $0B     mit der Anfangsadresse auf
F8F2:  85 AC     STA  * $AC     den System Kassettenpuffer
```

```
F8F4:  84 AD      STY  * $AD     ($0B00) initialisieren
F8F6:  20 D5 F9   JSR  $F9D5     Lade Startsektor 01 00 in Kas-Puffer
F8F9:  A2 00      LDX  # $00     Schleifen und Displ.-Zeiger löschen
F8FB:  BD 00 0B   LDA  $0B00,X   Überprüfe die ersten drei Bytes des
F8FE:  DD C4 E2   CMP  $E2C4,X   von der Diskette gelesenen Start-
F901:  D0 E8      BNE  $F8EB     sektors im Kassettenpuffer auf den
F903:  E8         INX            Autostart Code (<C> <B> <M>). Wurde
F904:  E0 03      CPX  # $03     dieser Code gefunden, dann handelt
F906:  90 F3      BCC  $F8FB     es sich um ein BOOT-Programm
F908:  20 17 FA   JSR  $FA17     Kernal PRIMM: Zeichenkette ausgeben

*****************************    Kernal Konstante für BOOTING Meldung

F90B:  0D 42 4F 4F 54 49 4E 47   <Cr> <B> <O> <O> <T> <I> <N> <G>
F913:  20 00                     <Blank>

*****************************    Zeiger und Boot-Nachricht festlegen

F915:  BD 00 0B   LDA  $0B00,X   Hole 4 Adreßladezeiger aus dem
F918:  95 A9      STA  * $A9,X   BOOT-Sektor ab Adresse $0B03 und
F91A:  E8         INX            initialisiere damit die 2 Z-Page
F91B:  E0 07      CPX  # $07     Adreßzeiger in $AC-$AD / $AE-$AF
F91D:  90 F6      BCC  $F915     Schleifen, bis Zeiger geladen sind
F91F:  BD 00 0B   LDA  $0B00,X   Hole Ausgabezeichen aus Kas.-Puffer
F922:  F0 06      BEQ  $F92A     Der Wert $00 ist das Endekennzeichen
F924:  20 D2 FF   JSR  $FFD2     Kernal BSOUT: Ein Zeichen ausgeben
F927:  E8         INX            Displ. auf Kas-Puffer um 1 erhöhen
F928:  D0 F5      BNE  $F91F     Unbed. Sprung zur Zeichenausgabe
F92A:  86 9E      STX  * $9E     Displ. auf Kassettenpuffer sichern
F92C:  20 17 FA   JSR  $FA17     Kernal PRIMM: Zeichenkette ausgeben

*****************************    BOOTING Message Konstante

F92F:  2E 2E 2E 0D 00            <.> <.> <.> <Cr>

*****************************    BOOT Routine nach Vorlauf

F934:  A5 AE      LDA  * $AE     Bank Zeiger aus BOOT-Sektor in Bank
F936:  85 C6      STA  * $C6     Zeiger für STASH Routine kopieren
F938:  A5 AF      LDA  * $AF     Hole Zähler f. Anzahl d. BOOT-Blöcke
F93A:  F0 09      BEQ  $F945     Alle BOOT-Blöcke gelesen, dann Exit
F93C:  C6 AF      DEC  * $AF     Bootblockzähler um 1 vermindern
F93E:  20 B3 F9   JSR  $F9B3     Nächsten Tr./Se. von Floppy laden
F941:  E6 AD      INC  * $AD     Ladeadresse Hi-Wert um 1 erhöhen
F943:  D0 F3      BNE  $F938     Sprung, um nächsten Block zu laden
F945:  20 8B F9   JSR  $F98B     Floppy nach BOOT initialisieren
```

```
F948:  A6 9E     LDX  * $9E      Displacement auf Kassettenpuffer
F94A:  2C        .Byte  $2C      Skip nach $F94D
F94B:  E6 9F     INC  * $9F      Dateinamen-Längenzähler um 1 erhöhen
F94D:  E8        INX             Displ.auf Zeichen nach 0-Code setzen
F94E:  BD 00 0B  LDA  $0B00,X    Hole Zeichen nach 0-Code (Dateiname)
F951:  D0 F8     BNE  $F94B      Nicht Null (Trenncode), weiter lesen
F953:  E8        INX             Displ.auf Zeichen nach 0-Code setzen
F954:  86 04     STX  * $04      und im PC-Lo Zeiger ablegen
F956:  A6 9E     LDX  * $9E      Displ. auf Zeichen vor Dateinamen
F958:  A9 3A     LDA  # $3A      0-Trenncode durch <:> ersetzen
F95A:  9D 00 0B  STA  $0B00,X    und dem Dateinamen voranstellen
F95D:  CA        DEX             Displ. auf Zeichen vor <:> setzen
F95E:  A5 BF     LDA  * $BF      ASCII-Zeichen der Laufwerksangabe
F960:  9D 00 0B  STA  $0B00,X    dem Dateinamen voranstellen <0:xxxx>
F963:  86 9E     STX  * $9E      Lo-Adresse des Dateinamens sichern
F965:  A6 9F     LDX  * $9F      Hole gesicherte Länge d. Dateinamens
F967:  F0 15     BEQ  $F97E      Kein Dateiname vorhanden, dann Skip
F969:  E8        INX             Dateinamen-Längenzeiger um 2 erhöhen
F96A:  E8        INX             da <0:> Zeichen mitgezählt werden
F96B:  8A        TXA             Länge des Dateinamens in A kopieren
F96C:  A6 9E     LDX  * $9E      Adresse Lo des Dateinamens holen
F96E:  A0 0B     LDY  # $0B      Adresse Hi des Dateinamens setzen
F970:  20 31 F7  JSR  $F731      Routine SETNAM: Setze Dateinamen
F973:  A9 00     LDA  # $00      Akku und X-Register für die SETBNK-
F975:  AA        TAX             Routine mit $00 initialisieren
F976:  20 3F F7  JSR  $F73F      Routine SETBNK: Bank für LSV+Datnam.
F979:  A9 00     LDA  # $00      Akku als "LOAD" Kennzeichen setzen
F97B:  20 69 F2  JSR  $F269      Sprung zum Kernal LOAD-Vektor
F97E:  A9 0B     LDA  # $0B      Den Z-Page Speicher für den PC-Hi
F980:  85 03     STA  * $03      auf $0B (Kassettenpuffer) setzen
F982:  A9 0F     LDA  # $0F      Den Z-Page Bank Zeiger auf den Wert
F984:  85 02     STA  * $02      $0F (System-Rom) setzen
F986:  20 CD 02  JSR  $02CD      JSRFAR Rout.: JSR beliebig.Bank +RTS
F989:  18        CLC             Carry für Kennzeichen OK löschen
F98A:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   Floppy Initialisierung für BOOTING

F98B:  08        PHP             Prozessor-Status auf Stack retten
F98C:  48        PHA             Akku Inhalt auf dem Stack retten
F98D:  20 CC FF  JSR  $FFCC      Kernal CLRCH: E/A Kanäle rücksetzen
F990:  A9 0D     LDA  # $0D      Logische Dateinummer (13) schließen
F992:  18        CLC             Carry auf "alles OK" setzen
F993:  20 C3 FF  JSR  $FFC3      Kernal CLOSE: Datei schließen
F996:  A2 00     LDX  # $00      Logische Datei (0) a. Ausgabe setzen
F998:  20 C9 FF  JSR  $FFC9      Kernal CKOUT: Ausgabekanal setzen
```

```
F99B:  B0 0A     BCS  $F9A7     Wenn Fehler, dann wieder schließen
F99D:  A9 55     LDA  # $55     Akku mit Zeichen <U> laden
F99F:  20 D2 FF  JSR  $FFD2     Kernal BSOUT: Ein Zeichen ausgeben
F9A2:  A9 49     LDA  # $49     Akku mit Zeichen <I> laden
F9A4:  20 D2 FF  JSR  $FFD2     Kernal BSOUT: Ein Zeichen ausgeben
F9A7:  20 CC FF  JSR  $FFCC     Kernal CLRCH: E/A Kanäle rücksetzen
F9AA:  A9 00     LDA  # $00     Logische Dateinummer (0) schließen
F9AC:  38        SEC            Carry auf "Fehler gefunden" setzen
F9AD:  20 C3 FF  JSR  $FFC3     Kernal CLOSE: Datei schließen
F9B0:  68        PLA            Akku Inhalt vom Stack zurückholen
F9B1:  28        PLP            Alten Prozessor-Status zurückholen
F9B2:  60        RTS            Rücksprung aus dem Unterprogramm

******************************  Track u. Sektor im DOS-Ausgabepuffer
                                neu setzen und Sektor laden

F9B3:  A6 C2     LDX  * $C2     Hole Sektor Nr. aus Z-Page Speicher
F9B5:  E8        INX            Sektor um 1 erhöhen
F9B6:  E0 15     CPX  # $15     Prüfe auf gültige Sektornummer
F9B8:  90 04     BCC  $F9BE     Sektor Nr. kleiner 21, dann alles OK
F9BA:  A2 00     LDX  # $00     Wert für Sektor Nr. Null laden
F9BC:  E6 C1     INC  * $C1     Track Nr. um 1 erhöhen
F9BE:  86 C2     STX  * $C2     Z-Page Sektor Nr. neu setzen
F9C0:  8A        TXA            Sektor Nr. in Akku kopieren und
F9C1:  20 FB F9  JSR  $F9FB     Sektor in 2 Byte ASCII umwandeln
F9C4:  8D 00 01  STA  $0100     Sektor Nr. Lo in DOS-Puffer ablegen
F9C7:  8E 01 01  STX  $0101     Sektor Nr. Hi in DOS-Puffer ablegen
F9CA:  A5 C1     LDA  * $C1     Lade Akku mit Track Nr. aus Z-Page
F9CC:  20 FB F9  JSR  $F9FB     Track in 2 Byte ASCII umwandeln
F9CF:  8D 03 01  STA  $0103     Track Nr. Lo in DOS-Puffer ablegen
F9D2:  8E 04 01  STX  $0104     Track Nr. Hi in DOS-Puffer ablegen
F9D5:  A2 00     LDX  # $00     Logische Datei #0 für CKOUT setzen
F9D7:  20 C9 FF  JSR  $FFC9     Kernal CKOUT: Ausgabekanal setzen
F9DA:  A2 0C     LDX  # $0C     13 Zeichen aus DOS-Puffer ausgeben
F9DC:  BD 00 01  LDA  $0100,X   Hole 1 Zeichen aus DOS Ausgabepuffer
F9DF:  20 D2 FF  JSR  $FFD2     Kernal BSOUT: Ein Zeichen ausgeben
F9E2:  CA        DEX            Schleifenzähler auf DOS Puffer -1
F9E3:  10 F7     BPL  $F9DC     Schleifen, bis 13 Zeichen ausgegeben
F9E5:  20 CC FF  JSR  $FFCC     Kernal CLRCH: E/A Kanäle rücksetzen
F9E8:  A2 0D     LDX  # $0D     Logische Datei (13) a.Eingabe setzen
F9EA:  20 C6 FF  JSR  $FFC6     Kernal CHKIN: Eingabekanal setzen
F9ED:  A0 00     LDY  # $00     Displ. für STASH Routine auf #0
F9EF:  20 CF FF  JSR  $FFCF     Kernal BASIN: Ein Zeichen einlesen
F9F2:  20 BC F7  JSR  $F7BC     STASH Routine für LSV Operationen
F9F5:  C8        INY            STASH Displ. Zeiger um 1 erhöhen
F9F6:  D0 F7     BNE  $F9EF     Schleifen,bis 256 Bytes gelesen sind
```

```
F9F8:  4C CC FF    JMP  $FFCC      Kernal CLRCH: E/A Kanäle rücksetzen

******************************     Akku Inhalt als 2 Byte ASCII aufbe-
                                   reiten (X=Hi, A=Lo) (nur bis #99)

F9FB:  A2 30       LDX  # $30      ASCII-Wert für Zeichen <0> nach X
F9FD:  38          SEC             Carry für Subtraktion setzen
F9FE:  E9 0A       SBC  # $0A      Dez. 10 vom Akku subtrahieren
FA00:  90 03       BCC  $FA05      Carry gelöscht, dann Unterlauf, Exit
FA02:  E8          INX             ASCII Hi Zeichen um 1 erhöhen
FA03:  B0 F9       BCS  $F9FE      Unbedingter Sprung
FA05:  69 3A       ADC  # $3A      Unterlauf abfangen,ASCII Lo erzeugen
FA07:  60          RTS             Rücksprung aus dem Unterprogramm

******************************     Kernal Konstante für BOOT-LOAD
                                   Ausgangswert für DOS-Ausgabepuffer

FA08:  30 30 20 31 30 20 30 20     <0> <0> < > <1> <0> < > <0> < >
FA10:  33 31 3A 31 55 49 23        <3> <1> <:> <1> <U> <I> <#>

******************************     Kernal Routine: PRIMM
                                   Den auf JSR folgenden Text ausgeben

FA17:  48          PHA             Akku Inhalt auf Stack sichern
FA18:  8A          TXA             Aktuellen X-Reg Inhalt über Akku
FA19:  48          PHA             auf den Stack sichern
FA1A:  98          TYA             Aktuellen Y-Reg Inhalt über Akku
FA1B:  48          PHA             auf den Stack sichern
FA1C:  A0 00       LDY  # $00      Displacement Zeiger mit $00 laden
FA1E:  BA          TSX             Stack Pointer Zeiger in X-Reg laden
FA1F:  FE 04 01    INC  $0104,X    Lo Byte der RTS Adresse im Stack +1
FA22:  D0 03       BNE  $FA27      Kein Überlauf, Skip Hi Abgleich
FA24:  FE 05 01    INC  $0105,X    Hi Byte der RTS Adresse im Stack +1
FA27:  BD 04 01    LDA  $0104,X    Lo Byte der RTS Adresse im Stack in
FA2A:  85 CE       STA  * $CE      Z-Page bringen (für nachindiz Adr.)
FA2C:  BD 05 01    LDA  $0105,X    Hi Byte der RTS Adresse im Stack in
FA2F:  85 CF       STA  * $CF      Z-Page bringen (für nachindiz Adr.)
FA31:  B1 CE       LDA  ($CE),Y    Byte aus RTS Adr + Y-Reg holen
FA33:  F0 05       BEQ  $FA3A      $00 = Ende KZ, dann Exit Routine
FA35:  20 D2 FF    JSR  $FFD2      Kernal BSOUT: Ein Zeichen ausgeben
FA38:  90 E4       BCC  $FA1E      Kein Fehler, dann nächstes Zeichen
FA3A:  68          PLA             Ein Byte vom Stack holen u. d. alten
FA3B:  A8          TAY             Inhalt des Y-Reg wiederherstellen
FA3C:  68          PLA             Ein Byte vom Stack holen u. d. alten
FA3D:  AA          TAX             Inhalt des X-Reg wiederherstellen
FA3E:  68          PLA             Alten Akku Inhalt wiederherstellen
```

```
FA3F:  60          RTS              Rücksprung aus dem Unterprogramm

*****************************       NMI Routine

FA40:  D8          CLD              Dezimale Betriebsart zurücksetzen
FA41:  A9 7F       LDA  # $7F       NMI Kennzeichen setzen
FA43:  8D 0D DD    STA  $DD0D       NMI Möglichkeit löschen
FA46:  AC 0D DD    LDY  $DD0D       Flags lesen und löschen
FA49:  30 14       BMI  $FA5F       Prüfe, ob RS-232 aktiv ist
FA4B:  20 3D F6    JSR  $F63D       Shift RUN/STOP Tastaurabfrage
FA4E:  20 E1 FF    JSR  $FFE1       Kernal STOP: Auf Stop Taste prüfen
FA51:  D0 0C       BNE  $FA5F       Nicht gedrückt, dann Skip I/O Init.
FA53:  20 56 E0    JSR  $E056       Standard Vektoren f. I/O+Interrupt
FA56:  20 09 E1    JSR  $E109       I/O initialisieren
FA59:  20 00 C0    JSR  $C000       I/O init. und Bildschirm löschen
FA5C:  6C 00 0A    JMP  ($0A00)     BASIC Warm-Start Einsprung   ($4003)
FA5F:  20 05 E8    JSR  $E805       Sprung zur NMI Routine für RS-232
FA62:  4C 33 FF    JMP  $FF33       Rücksprung in die IRQ Aufruf Routine

*****************************       IRQ Routine

FA65:  D8          CLD              Dezimale Betriebsart zurücksetzen
FA66:  20 24 C0    JSR  $C024       Einsprung in Editor IRQ Routine
FA69:  90 12       BCC  $FA7D       Bei Raster-Interrupt Exit IRQ
FA6B:  20 F8 F5    JSR  $F5F8       Rout. UDTIM: Setze interne 24h Uhr
FA6E:  20 D0 EE    JSR  $EED0       Prüfe Kassettenrekorder-Tastatur
FA71:  AD 0D DC    LDA  $DC0D       Hole CIA Interrupt Steuerregister
FA74:  AD 04 0A    LDA  $0A04       Hole System NMI/Reset Status Zeiger
FA77:  4A          LSR  A           Prüfe, ob Bit 0 gelöscht ist
FA78:  90 03       BCC  $FA7D       Ja, dann zurück zur IRQ Routine
FA7A:  20 06 40    JSR  $4006       BASIC IRQ Einsprung
FA7D:  4C 33 FF    JMP  $FF33       Rücksprung in die IRQ Aufruf Routine

*****************************       Tastatur Decodiertabelle 1a
                                    ASCII-Zeichensatz normal

FA80:  14 0D 1D 88 85 86 87 11
FA88:  33 57 41 34 5A 53 45 01
FA90:  35 52 44 36 43 46 54 58
FA98:  37 59 47 38 42 48 55 56
FAA0:  39 49 4A 30 4D 4B 4F 4E
FAA8:  2B 50 4C 2D 2E 3A 40 2C
FAB0:  5C 2A 3B 13 01 3D 5E 2F
FAB8:  31 5F 04 32 20 02 51 03
FAC0:  84 38 35 09 32 34 37 31
FAC8:  1B 2B 2D 0A 0D 36 39 33
```

```
FAD0:  08 30 2E 91 11 9D 1D FF
FAD8:  FF

******************************   Tastatur Decodiertabelle 2a
                                 ASCII-Zeichensatz mit Shift

FAD9:  94 8D 9D 8C 89 8A 8B 91
FAE1:  23 D7 C1 24 DA D3 C5 01
FAE9:  25 D2 C4 26 C3 C6 D4 D8
FAF1:  27 D9 C7 28 C2 C8 D5 D6
FAF9:  29 C9 CA 30 CD CB CF CE
FB01:  DB D0 CC DD 3E 5B BA 3C
FB09:  A9 C0 5D 93 01 3D DE 3F
FB11:  21 5F 04 22 A0 02 D1 83
FB19:  84 38 35 18 32 34 37 31
FB21:  1B 2B 2D 0A 8D 36 39 33
FB29:  08 30 2E 91 11 9D 1D FF
FB31:  FF

******************************   Tastatur Decodiertabelle 3a
                                 ASCII-Zeichensatz mit C=

FB32:  94 8D 9D 8C 89 8A 8B 91
FB3A:  96 B3 B0 97 AD AE B1 01
FB42:  98 B2 AC 99 BC BB A3 BD
FB4A:  9A B7 A5 9B BF B4 B8 BE
FB52:  29 A2 B5 30 A7 A1 B9 AA
FB5A:  A6 AF B6 DC 3E 5B A4 3C
FB62:  A8 DF 5D 93 01 3D DE 3F
FB6A:  81 5F 04 95 A0 02 AB 03
FB72:  84 38 35 18 32 34 37 31
FB7A:  1B 2B 2D 0A 8D 36 39 33
FB82:  08 30 2E 91 11 9D 1D FF
FB8A:  FF

******************************   Tastatur Decodiertabelle 4a
                                 ASCII-Zeichensatz mit CTRL

FB8B:  FF FF FF FF FF FF FF FF
FB93:  1C 17 01 9F 1A 13 05 FF
FB9B:  9C 12 04 1E 03 06 14 18
FBA3:  1F 19 07 9E 02 08 15 16
FBAB:  12 09 0A 92 0D 0B 0F 0E
FBB3:  FF 10 0C FF FF 1B 00 FF
FBBB:  1C FF 1D FF FF 1F 1E FF
FBC3:  90 06 FF 05 FF FF 11 FF
```

```
FBCB:  84 38 35 18 32 34 37 31
FBD3:  1B 2B 2D 0A 8D 36 39 33
FBDB:  08 30 2E 91 11 9D 1D FF
FBE3:  FF

******************************   Tastatur Decodiertabelle 5a
                                 ASCII-Zeichensatz mit ALT

FBE4:  14 0D 1D 88 85 86 87 11
FBEC:  33 D7 C1 34 DA D3 C5 01
FBF4:  35 D2 C4 36 C3 C6 D4 D8
FBFC:  37 D9 C7 38 C2 C8 D5 D6
FC04:  39 C9 CA 30 CD CB CF CE
FC0C:  2B D0 CC 2D 2E 3A 40 2C
FC14:  5C 2A 3B 13 01 3D 5E 2F
FC1C:  31 5F 04 32 20 02 51 03
FC24:  84 38 35 09 32 34 37 31
FC2C:  1B 2B 2D 0A 0D 36 39 33
FC34:  08 30 2E 91 11 9D 1D FF
FC3C:  FF

******************************   Freier Bereich

FC3D:  FF FF FF . . .
FC7D:  . . . FF FF FF

******************************

******************************   SID-Register und Edit-Zeiger löschen

FC80:  8D C5 0A   STA  $0AC5     System Akzent-Mode Flag löschen(A=0)
FC83:  8D 18 D4   STA  $D418     SID-Lautstärke Register löschen
FC86:  60         RTS            Rücksprung aus dem Unterprogramm

******************************   Einsprung in Kernal Routine: KEY

FC87:  2C C5 0A   BIT  $0AC5     Prüfe Bit 7 des Akzent-Mode Flags
FC8A:  30 37      BMI  $FCC3     Bit 7 gesetzt, Akzent zusammenbauen
FC8C:  A5 D3      LDA  * $D3     hole aktuelles SHIFT Muster in Akku
FC8E:  29 10      AND  # $10     Prüfe Bit 4 f. ASCII-DIN Umschaltung
FC90:  F0 0D      BEQ  $FC9F     Wenn ASCII Zeichensatz gewählt, Skip
FC92:  AD 3F 03   LDA  $033F     Prüfe, ob die Hi-Adresse der ersten
FC95:  C9 FD      CMP  # $FD     Decodiertabelle auf DIN-Satz zeigt
FC97:  F0 2A      BEQ  $FCC3     Ja, dann OK und Skip
FC99:  A9 34      LDA  # $34     X und A als Zeiger für Vektortabelle
FC9B:  A0 FE      LDY  # $FE     auf DIN-Decodiertabellensatz laden
```

```
FC9D:  D0 0B     BNE  $FCAA      Unbedingter Sprung in Laderoutine
FC9F:  AD 3F 03  LDA  $033F      Prüfe, ob die Hi-Adresse der ersten
FCA2:  C9 FA     CMP  # $FA      Decodiertabelle aus ASCII-Satz zeigt
FCA4:  F0 1D     BEQ  $FCC3      Ja, dann OK und Skip
FCA6:  A9 6F     LDA  # $6F      X und A als Zeiger für Vektortabelle
FCA8:  A0 C0     LDY  # $C0      auf ASCII-Decodiertabellensatz laden

******************************   Tabellensatzvektoren neu setzen

FCAA:  85 CC     STA  * $CC      Zeiger auf Vektortabelle Lo sichern
FCAC:  84 CD     STY  * $CD      Zeiger auf Vektortabelle Hi sichern
FCAE:  A0 0B     LDY  # $0B      Schleifenzähler, für 6 Vektoren
FCB0:  B1 CC     LDA  ($CC),Y    Hole Byte der ROM-Vektortabelle
FCB2:  99 3E 03  STA  $033E,Y    Sichere es in System-Vektortabelle
FCB5:  88        DEY             Vektor-Schleifenzähler vermindern
FCB6:  10 F8     BPL  $FCB0      Schleifen, bis 6 Vektoren kopiert
FCB8:  C8        INY             Y-Reg wieder auf Null hochzählen
FCB9:  8C C5 0A  STY  $0AC5      und das Akzent-Mode Flag löschen
FCBC:  08        PHP             Prozessor Status auf Stack sichern
FCBD:  78        SEI             Alle System Interrupts verhindern
FCBE:  20 0C CE  JSR  $CE0C      Kernal Routine: DLCHR
FCC1:  28        PLP             Prozessor Status zurückholen: CLI
FCC2:  60        RTS             Rücksprung aus dem Unterprogramm

******************************   Prüfe auf Akzent-Tasten und bilde
                                 zusammengesetzten Akzent

FCC3:  4C 5D C5  JMP  $C55D      Routine: Tastatur-Matrix abfragen
FCC6:  AE 3F 03  LDX  $033F      Prüfe, ob die Hi-Adresse der ersten
FCC9:  E0 FD     CPX  # $FD      Decodiertabelle auf DIN-Satz zeigt
FCCB:  D0 55     BNE  $FD22      Nein, Skip: Tastendruck speichern
FCCD:  AE C5 0A  LDX  $0AC5      Prüfe System Akzent-Mode Flag
FCD0:  30 50     BMI  $FD22      Bit 7 gesetzt, Tastendruck speichern
FCD2:  F0 1D     BEQ  $FCF1      Kein Akzent gesetzt, dann Skip
FCD4:  BC 45 FE  LDY  $FE45,X    Hole Wert aus Kombinationstabelle
FCD7:  CA        DEX             Tabellenwert um 1 vermindern
FCD8:  88        DEY             Displacement auf Tabelle - 1
FCD9:  48        PHA             Zeichencode auf Stapel sichern
FCDA:  98        TYA             Displacement auf Tab. in Akku holen
FCDB:  DD 45 FE  CMP  $FE45,X    Mit Kombinationstabelle vergleichen
FCDE:  68        PLA             Zeichencode vom Stapel zurückholen
FCDF:  90 08     BCC  $FCE9      Kombinationstabelle durchsucht, Skip
FCE1:  D9 4A FE  CMP  $FE4A,Y    Ist es ein Kombinationszeichen?
FCE4:  D0 F2     BNE  $FCD8      Kombinationstabelle weiter absuchen
FCE6:  B9 65 FE  LDA  $FE65,Y    Zusammengesetztes Zeichen aus Tab.
FCE9:  48        PHA             Zeichencode auf Stapel sichern
```

```
FCEA:  29 7F      AND  # $7F      Bit 7 ausblenden, keine RVS Zeichen
FCEC:  C9 20      CMP  # $20      Vgl. auf Space / Shift-Space
FCEE:  68         PLA             Zeichencode vom Stapel zurückholen
FCEF:  90 23      BCC  $FD14      Vgl. < $20: Steuerzeichen disable
FCF1:  A2 05      LDX  # $05      Schleifenzähler für Akzent-Tabelle
FCF3:  DD 3F FE   CMP  $FE3F,X    Vergleiche Zeichen m. Akzent-Tabelle
FCF6:  F0 03      BEQ  $FCFB      Zeichen in Tab. gefunden, Exit
FCF8:  CA         DEX             Schleifenzähler um 1 vermindern
FCF9:  D0 F8      BNE  $FCF3      Schleifen,bis alle Vgl. durchgeführt
FCFB:  8E C5 0A   STX  $0AC5      Displ. auf gefundenen Akzent sichern
FCFE:  E0 00      CPX  # $00      Displ. = #0, kein Akzent vorhanden
FD00:  F0 20      BEQ  $FD22      Wenn Null,dann Tastendruck speichern
FD02:  A8         TAY             Zeichencode in Y-Reg kopieren
FD03:  24 F6      BIT  * $F6      Prüfe, ob der Auto-Insert Modus
FD05:  30 0D      BMI  $FD14      eingeschaltet ist. Nein, dann RTS
FD07:  24 D7      BIT  * $D7      Prüfe 40/80 Zeichen Zeiger
FD09:  10 0A      BPL  $FD15      40 Zeichen Bildschirm aktiv, Skip
FD0B:  A2 0A      LDX  # $0A      X-Reg mit Nr. d. VDC Registers laden
FD0D:  20 DA CD   JSR  $CDDA      VDC-Register entsprechend auslesen
FD10:  29 40      AND  # $40      Prüfe aktuellen Cursor Modus
FD12:  D0 06      BNE  $FD1A      Wenn "Blink-Modus", Zeichen ausgeben
FD14:  60         RTS             Rücksprung aus dem Unterprogramm

******************************    Zusammengesetzten Akzent ausgeben

FD15:  AD 27 0A   LDA  $0A27      Prüfe Cursor On/Off Zeiger
FD18:  D0 FA      BNE  $FD14      Wert ungleich #0: Cursor disable;RTS
FD1A:  98         TYA             Zeichencode zurück in Akku holen
FD1B:  09 40      ORA  # $40      Bit 6 in Zeichencode einblenden
FD1D:  29 7F      AND  # $7F      Bit 7 ausblenden, keine RVS Zeichen
FD1F:  4C 2F CC   JMP  $CC2F      Zeichen an Cursorposition ausgeben
FD22:  A6 D3      LDX  * $D3      hole aktuelles SHIFT Muster in X-Reg
FD24:  A4 D5      LDY  * $D5      Flag für gedrückte Taste in Y-Reg
FD26:  6C 3C 03   JMP  ($033C)    Vektor:Tastendruck speichern ($C6AD)

******************************    Tastatur Decodiertabelle 1b
                                  DIN-Zeichensatz normal, Ctrl, Alt

FD29:  14 0D 1D 88 85 86 87 11
FD31:  33 57 41 34 59 53 45 01
FD39:  35 52 44 36 43 46 54 58
FD41:  37 5A 47 38 42 48 55 56
FD49:  39 49 4A 30 4D 4B 4F 4E
FD51:  BE 50 4C AF 2E BC BD 2C
FD59:  5B 2B BB 13 01 23 5D 2D
FD61:  31 3C 04 32 20 02 51 03
```

```
FD69:  84 38 35 09 32 34 37 31
FD71:  1B 2B 2D 0A 0D 36 39 33
FD79:  08 30 2E 91 11 9D 1D FF
FD81:  FF

******************************  Tastatur Decodiertabelle 2b
                                DIN-Zeichensatz mit Shift

FD82:  94 8D 9D 8C 89 8A 8B 91
FD8A:  40 D7 C1 24 D9 D3 C5 01
FD92:  25 D2 C4 26 C3 C6 D4 D8
FD9A:  2F DA C7 28 C2 C8 D5 D6
FDA2:  29 C9 CA 3D CD CB CF CE
FDAA:  3F D0 CC C0 3A DC DD 3B
FDB2:  5E 2A DB 93 01 27 5C 5F
FDBA:  21 3E 04 22 A0 02 D1 83
FDC2:  84 38 35 18 32 34 37 31
FDCA:  1B 2B 2D 0A 8D 36 39 33
FDD2:  08 30 2E 91 11 9D 1D FF
FDDA:  FF

******************************  Tastatur Decodiertabelle 3b
                                DIN-Zeichensatz mit C=

FDDB:  94 8D 9D 8C 89 8A 8B 91
FDE3:  96 A7 A8 97 A2 AA A3 01
FDEB:  98 A9 C4 99 C5 D3 CE A4
FDF3:  9A C2 DF 9B A1 C9 D6 D7
FDFB:  D1 C3 D5 C1 CB DA D8 CD
FE03:  AB D9 C8 BF BA CA B0 AC
FE0B:  AD A6 DB 93 01 DD DE B9
FE13:  81 B1 04 95 A0 02 A5 03
FE1B:  84 38 35 18 32 34 37 31
FE23:  1B 2B 2D 0A 8D 36 39 33
FE2B:  08 30 2E 91 11 9D 1D FF
FE33:  FF

******************************  Zeiger auf Tastatur Decodiertabellen
                                DIN-Zeichensatz Vektortabelle

FE34:  29 FD   ($FD29)          Tastatur Decodiertabelle 1b
FE36:  82 FD   ($FD82)          Tastatur Decodiertabelle 2b
FE38:  DB FD   ($FDDB)          Tastatur Decodiertabelle 3b
FE3A:  8B FB   ($FB8B)          Tastatur Decodiertabelle 4a
FE3C:  29 FD   ($FD29)          Tastatur Decodiertabelle 1b
```

```
FE3E:  29 FD   ($FD29)              Tastatur Decodiertabelle 1b

******************************      Tabelle der 3 Akzent Zeichen

FE40:  AF C0 BF 00 00               < > < > <^> (Letztes über: C=/< >)

******************************      Offset Tabelle auf mögliche Kombina-
                                    tionen zusammengesetzter Zeichen

FE45:  01 03 07 0C 0C 0C

******************************      Tabelle der möglichen Zeichen für
                                    e. zusammengesetztes Akzent-Zeichen

FE4B:  45 C0 41 45 55 AF 41 45      <E> < > <A> <E> <U> < > <A> <E>
FE53:  49 4F 55                     <I> <O> <U>

FE56:  FF FF FF FF FF FF FF FF      Füllwerte; nicht genutzt
FE5E:  FF FF FF FF FF FF FF FF

******************************      Tabelle der zusammengesetzten
                                    Akzent-Zeichen

FE64:  AC BF B2 AE B3 BF B4 B5      <e> <^> <a> <e> <u> <^> <a> <e>
FE6C:  B6 B7 B8                     <i> <o> <u>

FE71:  FF FF FF FF FF FF FF FF      Füllwerte; nicht genutzt
FE79:  FF FF FF . . .
FEFD:  . . . FF FF FF

******************************      Kopie der Configuration Register

FF00:  00          .Byte  $00       Configuration Register (CR)
FF01:  3F          .Byte  $3F       Load Configuration Register A (LCRA)
FF02:  7F          .Byte  $7F       Load Configuration Register B (LCRB)
FF03:  01          .Byte  $01       Load Configuration Register C (LCRC)
FF04:  41          .Byte  $41       Load Configuration Register D (LCRD)

******************************      Kernal NMI Routine

FF05:  78          SEI              Alle System Interrupts verhindern
FF06:  48          PHA              Akku Inhalt auf Stack sichern
FF07:  8A          TXA              Aktuellen X-Reg Inhalt über Akku
FF08:  48          PHA              auf dem Stack sichern
FF09:  98          TYA              Aktuellen Y-Reg Inhalt über Akku
FF0A:  48          PHA              auf dem Stack sichern
```

```
FF0B:  AD 00 FF   LDA   $FF00       Configuration Register in Akku holen
FF0E:  48         PHA               Config. Wert auf Stack sichern
FF0F:  A9 00      LDA   # $00       Configuration Register mit $00 laden
FF11:  8D 00 FF   STA   $FF00       und alle System ROMs einschalten
FF14:  6C 18 03   JMP   ($0318)     Vektor zeigt auf NMI Routine ($FA40)

*****************************       Kernal IRQ Routine

FF17:  48         PHA               Akku Inhalt auf Stack sichern
FF18:  8A         TXA               Aktuellen X-Reg Inhalt über Akku
FF19:  48         PHA               auf dem Stack sichern
FF1A:  98         TYA               Aktuellen Y-Reg Inhalt über Akku
FF1B:  48         PHA               auf dem Stack sichern
FF1C:  AD 00 FF   LDA   $FF00       Configuration Register in Akku holen
FF1F:  48         PHA               Config. Wert auf Stack sichern
FF20:  A9 00      LDA   # $00       Configuration Register mit $00 laden
FF22:  8D 00 FF   STA   $FF00       und alle System ROMs einschalten
FF25:  BA         TSX               Stack Pointer Zeig. in X-Reg bringen
FF26:  BD 05 01   LDA   $0105,X     Das vor dem Interrupt gesicherte CPU
FF29:  29 10      AND   # $10       Status Byte holen + Break Bit testen
FF2B:  F0 03      BEQ   $FF30       Kein Break aufgetreten,normal weiter
FF2D:  6C 16 03   JMP   ($0316)     Vektor zeigt auf BRK Routine ($B003)
FF30:  6C 14 03   JMP   ($0314)     Vektor zeigt auf IRQ Routine ($FA65)
FF33:  68         PLA               Alten Config. Wert vom Stack holen +
FF34:  8D 00 FF   STA   $FF00       gewählte Config. wiederherstellen
FF37:  68         PLA               Ein Byte vom Stack holen u. d. alten
FF38:  A8         TAY               Inhalt des Y-Reg wiederherstellen
FF39:  68         PLA               Ein Byte vom Stack holen u. d. alten
FF3A:  AA         TAX               Inhalt des X-Reg wiederherstellen
FF3B:  68         PLA               Alten Akku Inhalt wiederherstellen
FF3C:  40         RTI               Rücksprung aus der Interrupt Routine

*****************************       Kernal RESET Routine

FF3D:  A9 00      LDA   # $00       Configuration Register mit $00 laden
FF3F:  8D 00 FF   STA   $FF00       und alle System ROMs einschalten
FF42:  4C 00 E0   JMP   $E000       Reset Einsprung

*****************************       Kernal Vektor und Sprungtabelle

FF45:  FF         .Byte  $FF
FF46:  FF         .Byte  $FF

FF47:  4C FB E5   JMP   $E5FB       Zeiger auf Kernal FSTMOD    Routine
```

```
FF4A:  4C 3D F2   JMP  $F23D      Zeiger auf Kernal EAINIT    Routine

FF4D:  4C 4B E2   JMP  $E24B      Zeiger auf Kernal C64MODE   Routine

FF50:  4C A5 F7   JMP  $F7A5      Zeiger auf Kernal DMA-CALL  Routine

FF53:  4C 90 F8   JMP  $F890      Zeiger auf Kernal BOOT-CALL Routine

FF56:  4C 67 F8   JMP  $F867      Zeiger auf Kernal PHOENIX   Routine

FF59:  4C 9D F7   JMP  $F79D      Rout. LKUPLA: Suche in Tab. nach LFN

FF5C:  4C 86 F7   JMP  $F786      Rout. LKUPSA: Suche in Tab. nach SA

FF5F:  4C 2A C0   JMP  $C02A      Zeiger auf Kernal SWAPPER   Routine

FF62:  4C 27 C0   JMP  $C027      Zeiger auf Kernal DLCHR     Routine

FF65:  4C 21 C0   JMP  $C021      Zeiger auf Kernal PFKEY     Routine

FF68:  4C 3F F7   JMP  $F73F      Rout. SETBNK: Bank f. LSV+Dateinamen

FF6B:  4C EC F7   JMP  $F7EC      Zeiger auf Kernal GETCFG    Routine

FF6E:  4C CD 02   JMP  $02CD      Zeiger auf Kernal JSRFAR    Routine

FF71:  4C E3 02   JMP  $02E3      Zeiger auf Kernal JMPFAR    Routine

FF74:  4C D0 F7   JMP  $F7D0      Rout. INDFET: LDA(fetvec),Y bel.Bank

FF77:  4C DA F7   JMP  $F7DA      Rout. INDSTA: STA(stavec),Y bel.Bank

FF7A:  4C E3 F7   JMP  $F7E3      Rout. INDCMP: CMP(cmpvec),Y bel.Bank

FF7D:  4C 17 FA   JMP  $FA17      Zeiger auf Kernal PRIMM     Routine

FF80:  00         .Byte  $00

FF81:  4C 00 C0   JMP  $C000      Zeiger auf Kernal CINT      Routine

FF84:  4C 09 E1   JMP  $E109      Zeiger auf Kernal IOINIT    Routine

FF87:  4C 93 E0   JMP  $E093      Zeiger auf Kernal RAMTAS    Routine

FF8A:  4C 56 E0   JMP  $E056      Zeiger auf Kernal RESTOR    Routine
```

```
FF8D:  4C 5B E0   JMP  $E05B     Zeiger auf Kernal VECTOR    Routine
FF90:  4C 5C F7   JMP  $F75C     Zeiger auf Kernal SETMSG    Routine
FF93:  4C D2 E4   JMP  $E4D2     Routine SECND: Sekundäradr. f.LISTN
FF96:  4C E0 E4   JMP  $E4E0     Routine TKSA: Sekundäradr. für TALK
FF99:  4C 63 F7   JMP  $F763     Zeiger auf Kernal MEMTOP    Routine
FF9C:  4C 72 F7   JMP  $F772     Zeiger auf Kernal MEMBOT    Routine
FF9F:  4C 12 C0   JMP  $C012     Zeiger auf Kernal KEY       Routine
FFA2:  4C 5F F7   JMP  $F75F     Zeiger auf Kernal SETTMO    Routine
FFA5:  4C 3E E4   JMP  $E43E     Zeiger auf Kernal ACPTR     Routine
FFA8:  4C 03 E5   JMP  $E503     Zeiger auf Kernal CIOUT     Routine
FFAB:  4C 15 E5   JMP  $E515     Routine UNTLK: UNTLK Bef.an ser. Bus
FFAE:  4C 26 E5   JMP  $E526     Routine UNLSN: UNLSN Bef.an ser. Bus
FFB1:  4C 3E E3   JMP  $E33E     Routine LISTN: LISTN Bef.an ser. Bus
FFB4:  4C 3B E3   JMP  $E33B     Routine TALK: TALK Befehl an ser.Bus
FFB7:  4C 44 F7   JMP  $F744     Zeiger auf Kernal READST    Routine
FFBA:  4C 38 F7   JMP  $F738     Routine SETLFS: Setze Dateiparameter
FFBD:  4C 31 F7   JMP  $F731     Routine SETNAM: Setze Dateinamen
FFC0:  6C 1A 03   JMP  ($031A)   Vektor zeigt auf OPEN  Routine $EFBD
FFC3:  6C 1C 03   JMP  ($031C)   Vektor zeigt auf CLOSE Routine $F188
FFC6:  6C 1E 03   JMP  ($031E)   Vektor zeigt auf CHKIN Routine $F106
FFC9:  6C 20 03   JMP  ($0320)   Vektor zeigt auf CKOUT Routine $F14C
FFCC:  6C 22 03   JMP  ($0322)   Vektor zeigt auf CLRCH Routine $F226
FFCF:  6C 24 03   JMP  ($0324)   Vektor zeigt auf BASIN Routine $EF06
```

```
FFD2:  6C 26 03   JMP  ($0326)   Vektor zeigt auf BSOUT Routine $EF79

FFD5:  4C 65 F2   JMP  $F265     Routine LOADSP: Datei einladen

FFD8:  4C 3E F5   JMP  $F53E     Routine SAVESP: Datei abspeichern

FFDB:  4C 65 F6   JMP  $F665     Zeiger auf Kernal SETTIM    Routine

FFDE:  4C 5E F6   JMP  $F65E     Zeiger auf Kernal RDTIM     Routine

FFE1:  6C 28 03   JMP  ($0328)   Vektor zeigt auf STOP Routine  $F66E

FFE4:  6C 2A 03   JMP  ($032A)   Vektor zeigt auf GETIN Routine $EEEB

FFE7:  6C 2C 03   JMP  ($032C)   Vektor zeigt auf CLALL Routine $F222

FFEA:  4C F8 F5   JMP  $F5F8     Rout. UDTIM: Setze interne 24h Uhr

FFED:  4C 0F C0   JMP  $C00F     Zeiger auf Kernal SCRORG    Routine

FFF0:  4C 18 C0   JMP  $C018     Zeiger auf Kernal PLOT      Routine

FFF3:  4C 81 F7   JMP  $F781     Zeiger auf Kernal IOBASE    Routine

FFF6:  FF         .Byte  $FF
FFF7:  FF         .Byte  $FF

FFF8:  24 E2      ($E224)        C128Mode Vektor

FFFA:  05 FF      ($FF05)        NMI Vektor

FFFC:  3D FF      ($FF3D)        Reset Vektor

FFFE:  17 FF      ($FF17)        IRQ Vektor
```

9. Das BASIC-ROM

9.1 Allgemeines

Nach dem BASIC 1.0 des schon legendären PET wurden die Versionen BASIC 2.0 und BASIC 4.0 entwickelt. Mit Computern wie dem C16 kam dann das BASIC 3.5 mit speziellen Grafik- und Soundbefehlen auf den Markt. Nun sind wir bei den Commodore-Rechnern am BASIC-Höhepunkt angelangt, den sich dieses Teil des Buches zum Thema macht:

Der Commodore 128 und sein BASIC 7.0.

Dieses Kapitel soll Ihnen vor allem als Nachschlagewerk dabei helfen, speziell von Assembler aus alle Möglichkeiten des BASIC 7.0 zu nutzen. Da das BASIC 7.0 aber sehr ausgiebig von Betriebssystem und Hardware des Commodore 128 Gebrauch macht, sollten Sie Grundkenntnisse des Betriebssystems und der Hardware haben, damit Sie die Informationen in diesem Kapitel sinnvoll verwerten können. Die nötigen Informationen über das Betriebssystem finden Sie in den Kapiteln 7 und 8.

Stören Sie sich nicht an Querverweisen innerhalb des ROM-Listings. Das BASIC 7.0 ist so vielfältig, daß Sie auch für eine einzelne Sache meistens eine Fünf-Finger-Lesetechnik entwickeln müssen. Das bedeutet im Klartext: Ein Finger im aktuellen Kapitel, ein Finger im ROM-Listing, ein Finger in der Zeropage, zwei Finger in weiteren Kapiteln und die andere Hand an Ihrem Commodore 128. Lesen Sie dieses Buch wenn möglich immer mit Ihrem Commodore 128 in Reichweite zum Ausprobieren.

Falls Sie an dieser Stelle ein Kapitel mit Tabellen der Befehle, Fehlermeldungen, Operatoren etc. vermissen, dann hat das einen Grund: Sie finden diese Tabellen in gleichwertiger Form an den entsprechenden Stellen im ROM-Listing. Unnötige Wiederholungen wurden in Ihrem Interesse bewußt vermieden.

Lassen Sie uns jetzt auch gleich einige Vereinbarungen über die verwendeten Darstellungsformen treffen. Wird von Adressen

oder Adreßwerten gesprochen, dann ist das erste Byte - soweit nicht anders angegeben - immer das Low-Byte. Ist ein Zwei-Byte-Wert in zwei Prozessorregistern gespeichert, so steht ein Schrägstrich zwischen den Registernamen. Auch hier ist das erste Byte das Low-Byte. Ist ein Adreßwert in zwei aufeinanderfolgenden Speicherzellen gemeint, dann steht die Adresse der ersten Speicherzelle in Klammern. Es ist also genauso, wie bei der nachindizierten Adressierung des 6502-Assemblers.

9.2 Nicht vorhandene Befehle

Commodore hatte anscheinend einige Befehle geplant, die hinterher doch nicht programmiert worden sind. Das verwundert doch sehr, da noch viel freier Platz im BASIC-ROM vorhanden ist. Zu diesen Befehlen gehören zum Beispiel:

QUIT, KEY ON, KEY OFF

Wenn Sie die entsprechenden Vektoren der Kommandos verbiegen, so können Sie eigene Maschinenprogramme mit diesen Kommandos aufrufen.

9.3 Die Variablen

Zur Verwaltung der Variablen gibt es ebenfalls etwas zu sagen. Wenn Sie Ihr BASIC-Programm ändern, so werden die bestehenden Variablen nicht gelöscht. Auch nach einer Fehlermeldung sind alle Variablenwerte noch vorhanden. Das Originalhandbuch von Commodore ist in diesem Punkt (Kap. 3-7) bedauerlicherweise falsch. Dieser Unterschied zum C64 und den anderen Commodore-Computern kann viel Kopfzerbrechen bereiten, erinnert man sich an diese Tatsache einmal nicht. Sollten Sie jetzt denken, daß Sie Ihr Programm nach einer Fehlermeldung einfach ändern und mit GOTO weiterlaufen lassen können, so ist das trotzdem nicht möglich. Das BASIC 7.0 löscht nach Änderungen zwar nicht die Variablen, aber alle Parameter für FOR-NEXT-Schleifen, GOSUB-Befehle und alle anderen

Sprung- oder Schleifenbefehle gehen verloren (der BASIC-Stack also).

9.4 Die Speicheraufteilung

Wie Sie wissen, hat Ihr Commodore 128 zwei RAM-Bänke zu je 64 KByte. Diese werden vom BASIC 7.0 fast vollständig ausgenutzt. Das BASIC 7.0 trennt strikt nach Programm und Variablen. Programme sind ausschließlich in Bank 0 vorhanden, Variablen dagegen in Bank 1.

Der Grafikspeicher fällt etwas aus dem Rahmen, da er nur dann reserviert wird, wenn wirklich Grafik dargestellt werden soll. Wenn Sie den GRAPHIC-Befehl zum ersten Mal benutzen, wird Ihr BASIC-Programm um 9 KByte in der Bank 0 nach oben verschoben. Alle Adressen auf dem Prozessorstack oder anderswo, die in das Programm zeigen, werden durch eine sehr aufwendige Routine in der Reservierung ab $9F4F angepaßt. Ein BASIC-Programm liegt dann nicht mehr ab $1C00 im Speicher sondern, ab $4000. Der Speicher von $1C00 bis $1FFF wird dann als Farb-RAM für die Grafik benutzt. Die Grafik selber belegt die restlichen 8 KByte von $2000 bis $4000. Sobald Sie den GRAPHIC CLR-Befehl benutzen, wird der Grafikspeicher gelöscht und das Programm wieder nach $1C00 verschoben. Nicht nur die Verwaltung der Variablen ist also *dynamisch*, auch die Programmbank wird so verwaltet.

In der Variablenbank 1 werden am Anfang die einfachen Variablen gespeichert. Direkt dahinter sind die Feldvariablen (*Arrays*) zu finden. Der folgende Speicher ist frei. Vom Ende des Variablenspeichers abwärts werden die Stringinhalte gespeichert. Damit der Interpreter sich zurechtfindet, verwaltet er die Zeiger ($2D) bis ($35) für Programm und Daten. Das Programmende ist in ($1210) und das Ende des verfügbaren Programmspeichers in ($1212) gespeichert.

9.5 Datenformate des BASIC 7.0

9.5.1 Das Programmzeilenformat

Eine Programmzeile beginnt im Speicher mit einer Verkettungsadresse (Linkadresse) auf die nächste Zeile. Die Linkadresse besteht aus zwei Byte. Sie erleichtert das schnelle Auffinden von einzelnen Zeilen. Ist die Linkadresse gleich Null, dann ist das Programmende erreicht. Es folgen dann keine Daten mehr.

Die folgenden zwei Byte geben die Zeilennummer der Zeile an.

Nun folgt der Programmtext der Zeile. Befehle sind als Token abgelegt. Eine Programmzeile kann theoretisch maximal 250 Zeichen Text beinhalten, ohne den Interpreter zu verwirren. Praktisch ist diese Zahl aber durch die Länge des Eingabepuffers auf 160 beschränkt.

Ein Null-Byte $00 schließt jede Programmzeile ab. Das Null-Byte hat aber noch eine zweite Funktion. Es zeigt auch an, daß danach wieder eine Verkettungsadresse folgt. Demnach zeigt es gleichzeitig auch den Anfang einer Programmzeile an. Das läßt sich sehr anschaulich zeigen. Tippen Sie einfach einmal zwei Zeilen Programm irgendeiner Art ein. Schreiben Sie nun mit dem Monitor in die Speicherzelle $1C00 einen Wert ungleich null. Geben Sie nun

RUN

ein. Sie erhalten eine Fehlermeldung vom Interpreter, der das Null-Byte in $1C00 nicht gefunden hat; er konnte also nicht den Anfang einer neuen Programmzeile oder einen Doppelpunkt als Trennzeichen finden und hat dies gemeldet. Auch das Eintippen von

NEW

löst das Problem nicht. Erst ein Schreiben von $00 an die Adresse $1C00 läßt die Fehlermeldung wieder verschwinden.

9.5.2 Das Format von Realzahlen

Das Format von reellen Zahlen ist etwas kompliziert. Es gibt nämlich zwei Möglichkeiten der Darstellung: Das Speicherformat und das sogenannte FAC-Format.

Das Speicherformat einer Realzahl belegt fünf Byte. Im ersten Byte ist der Exponent gespeichert. Ist die Zahl Null, so ist der Exponent auch Null. Ist die Zahl aber nicht Null, dann ist der Grundwert des Exponenten immer $80, also dezimal 128. Man hat deshalb nach *unten* und nach *oben* jeweils 127 Werte Platz, was einen Exponentenbereich von -127 bis +127 ermöglicht. In den restlichen vier Byte ist der Realteil der Zahl - die sogenannte Mantisse - gespeichert. Das höchstwertige Bit des Realteils, das immer eins wäre, ist nicht abgespeichert. Es wird von den Arithmetikroutinen aus dem Realteil herausgeschoben. Der Exponent wird entsprechend angepaßt. Dieser Vorgang heißt *Normalisierung*. Das Vorzeichen der Zahl belegt Bit 7 des zweiten Bytes im Speicherformat.

Das FAC-Format (*Floating-point-ACumulator*), das seinen Namen von der englischen Bezeichnung der Rechenregister hat, ist etwas anders aufgebaut als das Speicherformat. Es besteht aus sechs Byte. Im ersten Byte ist der Exponent gespeichert. Das zweite Byte ist fast identisch mit dem des Speicherformats. Das Vorzeichen in Bit 7 ist aber immer gesetzt. Byte drei bis fünf entsprechen denen des Speicherformats. Byte sechs enthält ausschließlich das Vorzeichen aus Bit 2 des Speicherformats.

Wichtig für Sie ist normalerweise nur das Speicherformat, da das FAC-Format automatisch vom Interpreter erzeugt und nur zum Rechnen benutzt wird.

9.5.3 Das Format von Integerzahlen

Die Integerzahlen belegen jeweils zwei Byte. Sie sind im High/Low-Format abgespeichert. Das höherwertige Byte kommt also immer zuerst! Beachten Sie diesen Unterschied zur Assemblerprogrammierung des 6502 unbedingt. Die Integerzahlen sind in Zweierkomplementform gespeichert. Das bedeutet, daß negative Zahlen gebildet werden, indem man die

normalen positiven invertiert und eins aufaddiert. Das Vorzeichen einer Zahl läßt sich also im Bit 7 des High-Bytes erkennen. Einige Beispiele hierzu:

```
    +5 = $00 $05
    -5 = $FF $FB
+32767 = $7F $FF
-32768 = $80 $00
```

Integerzahlen haben also einen Wertebereich von -32768 bis +32767. Das BASIC 7.0 besitzt aber leider keine Integerarithmetik. Jede Integerzahl wird zuerst in eine Realzahl gewandelt. Nach einer Rechenoperation wird das reelle Ergebnis wieder in das Integerformat konvertiert. Sie sehen, die Benutzung von Integerwerten bringt keinerlei Vorteile.

9.5.4 Das Format der Variablennamen

Jeder Variablenname hat zwei gültige Zeichen. Ist kein zweites Zeichen angegeben, so wird stattdessen ein Null-Byte verwendet. Der Variablentyp ist im Bit 7 der beiden Zeichen angegeben. Dabei gelten folgende Zuordnungen:

```
   Realvariable: Zeichen 1         Zeichen 2
       Funktion: Zeichen 1 + $80   Zeichen 2
 Stringvariable: Zeichen 1         Zeichen 2 + $80
Integervariable: Zeichen 1 + $80   Zeichen 2 + $80
```

Wie Sie sehen, werden auch Funktionen als Variable im Speicher abgelegt. Deshalb wird das Format eines Funktionsnamens hier ebenfalls beschrieben.

9.5.5 Das Format von Realvariablen

Realvariablen belegen im Speicher sieben Byte. Der Name ist wie in Kapitel 9.5.4 beschrieben kodiert. Die eigentliche Realzahl ist im Speicherformat abgelegt.

Eine Realvariable hat folgendes Format im Speicher:

Byte 1: Name 1
Byte 2: Name 2
Byte 3: Höchstwertiges Byte der Mantisse
Byte 4: Mantissenbyte
Byte 5: Mantissenbyte
Byte 6: Mantissenbyte
Byte 7: Niederwertiges Byte der Mantisse

Sie müssen sich nicht um das Speicherformat der Realzahl kümmern, wollen Sie nicht in Maschinensprache direkt Einfluß hierauf nehmen. Die ROM-Routinen, die einen Wert aus dem Speicher in ein Rechenregister (FAC) des Interpreters kopieren, verarbeiten das Speicherformat automatisch.

9.5.6 Das Format von Funktionen

Auch Funktionen, die Sie mit dem BASIC-Befehl DEF FN definiert haben, werden wie normale Variablen im Speicher abgelegt (wir erwähnten dies bereits). Sie belegen ebenfalls sieben Byte und unterscheiden sich nur in der Kodierung des Namens von den anderen Variablen.

Eine Funktion hat folgendes Format im Speicher:

Byte 1: Name 1 + $80
Byte 2: Name 2
Byte 3: Zeiger auf Funktion Low
Byte 4: Zeiger auf Funktion High
Byte 5: Zeiger auf Funktionsvariable Low
Byte 6: Zeiger auf Funktionsvariable High
Byte 7: Unbenutzt

9.5.7 Das Format von Stringvariablen

Stringvariablen belegen im Speicher sieben Byte. Sie beinhalten nicht den eigentlichen String, sondern nur die Länge und die Adresse des Strings.

Eine Stringvariable hat folgendes Format im Speicher:

Byte 1: Name 1
Byte 2: Name 2 + $80
Byte 3: Stringlänge
Byte 4: Stringadresse Low
Byte 5: Stringadresse High
Byte 6: $00
Byte 7: $00

Zwei Byte werden also nicht benutzt, weswegen man möglichst viel in einem String speichern sollte. Die eigentlichen Strings werden vom Ende des Variablenspeichers ab abwärts angelegt. Am Ende jedes Strings werden noch zwei Zeigerbyte angelegt, der sogenannte *Trailer*. Er zeigt auf die Stringlänge: Byte 3 der Variablendefinition. Wird der String durch irgendeine Operation ungültig, so wird er nicht gelöscht. Im Low-Byte des Trailers wird die Stringlänge eingetragen und das High-Byte des Trailers wird einfach auf $FF gesetzt. Der alte String gilt damit als ungültig. Nun wird ein neuer Platz für den neu erstellten String reserviert. Die Vernichtung der alten Strings wird in der Garbage Collection erledigt (Kapitel 9.6).

9.5.8 Das Format von Integervariablen

Integervariablen belegen im Speicher sieben Byte und haben das folgende Format im Speicher:

Byte 1: Name 1 + $80
Byte 2: Name 2 + $80
Byte 3: High-Byte
Byte 4: Low-Byte
Byte 5: $00
Byte 6: $00
Byte 7: $00

Wie man sieht, ist jede Verwendung einer Integervariablen Verschwendung von drei Byte. Außerdem verlangsamen Integervariablen den Rechenvorgang des Interpreters nur, da er keine Integerarithmetik besitzt und jede ganze Zahl erst in eine Realzahl wandelt, bevor er sie verarbeitet.

9.5.9 Das Format von Feldern

Felder *(Arrays)* benötigen etwas mehr Verwaltungsaufwand als einfache Variablen. Da Feldvariablen nicht mit den einfachen Variablen gemischt sind, sondern im Speicher dahinter gespeichert werden, ist keine besondere Bezeichnung nötig. Ein Feld besteht aus einem Feldkopf *(Deskriptorblock)*, der die Eigenschaften und die Größe des Feldes beschreibt, sowie dem gesamten Feldinhalt. In den ersten zwei Byte finden Sie den Namen des Feldes. In den nächsten zwei Byte ist die Gesamtlänge des Feldes gespeichert. Der Interpreter benutzt diese Adreßangabe, um beim Suchen ein Feld schnell überspringen zu können. In Byte 5 ist die Anzahl der Felddimensionen abgespeichert. Ab Byte 6 finden Sie dann die Größen der einzelnen Dimensionen. Da man mehr als 255 Elemente pro Dimension benutzen darf, wird jede einzelne Größe als Adreßangabe gespeichert und belegt demnach zwei Byte. Die letzte Dimension, die im DIM-Befehl genannt worden ist, wird als erste angegeben. Sie müssen also auf die umgekehrte Reihenfolge achten. Der Feldinhalt schließt sich direkt an. Ist das Feld eindimensional, so werden die Elemente der Reihe nach abgelegt, bei mehrdimensionalen Feldern spaltenweise. Die Größe eines Elementes richtet sich nach dem Feldtyp. Ein Realelement belegt fünf Byte und ist im Speicherformat abgelegt. Ein Integerelement benötigt dagegen nur zwei Byte. Ein Stringelement braucht drei Byte, da nicht nur die Adresse sondern auch die Länge des Strings abgespeichert werden muß.

Wenn Sie statt einfachen Integervariablen Integerfelder benutzen sollten, dann sparen Sie erheblich an Speicherplatz. Denn auch 60 KByte Speicher für Variablen können sehr schnell belegt sein.

9.6 Die Garbage Collection

Was soll eine *Müllsammlung* in Ihrem Commodore 128? Das ist einfach zu erklären. Wird vom Interpreter für einen alten String ein neuer angelegt (beispielsweise bei Addition eines Zeichens), so wird der alte String nicht gelöscht. Durch einen Vermerk im Trailer wird er einfach für ungültig erklärt, der neue String bekommt einen eigenen Platz zugewiesen. Bei intensiver

Benutzung von Strings in Ihrem Programm kann es nun dazu kommen, daß der Variablenspeicher "bis zum Rand" gefüllt ist. Wenn der Interpreter dies feststellt, dann gibt er nicht einfach einen Fehler aus. Er versucht erst einmal den gesamten *Stringmüll*, der den Speicher füllt, zu sammeln und zu löschen.

Wie funktioniert nun die Garbage Collection? Der Interpreter durchsucht die Strings vom Ende des Speichers an. Anhand des Trailers kann er die Länge eines Strings sehr schnell finden und überspringt gültige Strings. Findet er aber einen ungültigen String, bei dem das High-Byte des Trailers $FF ist, so holt er sich aus dem Low-Byte die Länge des Strings, schiebt vom Anfang des Stringspeichers an alle anderen Strings nach oben und löscht auf diese Weise den ungültigen String. Nun werden werden alle Zeiger *(Deskriptoren)* auf die verschobenen Strings in den Variablen- und Feldbeschreibungen angepaßt. Danach wird von der aktuellen Position aus der nächste ungültige String gesucht und gelöscht, bis der Anfang des Stringspeichers erreicht ist. Durch die Übernahme des Trailers aus dem BASIC 4.0 in das BASIC 7.0 konnte die Stringverarbeitung gegenüber dem BASIC 2.0 des C64 so extrem beschleunigt werden, daß man sie kaum noch bemerkt.

9.7 Die Stacks

Wieso Stacks, der Prozessor hat doch nur einen Stapelspeicher, werden Sie sagen. Das stimmt schon, aber das BASIC 7.0 verwaltet noch einen eigenen Stack ab $0800 für Informationen über FOR-NEXT-Schleifen, GOSUB-Befehle und die anderen Sprung- und Schleifenbefehle. Der Prozessorstapel wird vom BASIC 7.0 auch benutzt, aber nur zur Auswertung von Ausdrücken. Beim C64 waren diese Informationen alle auf dem Prozessorstapel gespeichert. Da bei Ihrem Commodore 128 aber nun ein eigener Speicher für das BASIC 7.0 vorhanden ist, können Sie wesentlich mehr Schleifen und Unterprogrammaufrufe verwenden, ohne eine Fehlermeldung zu verursachen. Probieren sie dieses kleine Programm einmal im C64-Modus und dann wieder im C128-Modus aus:

```
10 a=0
20 a=a+1:printa:gosub20
```

Der Unterschied der Zahlen, die zuletzt ausgegeben werden, ist sicherlich interessant.

9.8 Interrupts im BASIC 7.0

Programmunterbrechungen werden im BASIC 7.0 durch den COLLISION-Befehl erlaubt. Das BASIC 7.0 hat eine eigene Interruptroutine, die vom Betriebssystem zusammen mit der Tastaturabfrage regelmäßig aufgerufen wird. Falls die Interruptlogik eingeschaltet ist, setzt diese Routine entsprechende Flags, falls zum Beispiel der Lichtgriffel benutzt worden ist. Der Interpreter testet bei jedem Befehlsaufruf, ob ein Flag gesetzt worden ist. Wenn dies passiert ist, dann rettet er die Arbeitswerte auf den Stapel, sperrt die Interruptlogik und ruft sich selbst - und damit das entsprechende Unterprogramm - mit dem GOSUB-Befehl rekursiv auf.

Eine etwas andere Art der Programmunterbrechung wird vom TRAP-Befehl unterstützt. Tritt im Programm ein Fehler auf, dann wird getestet, ob mit dem TRAP-Befehl eine Fehlerbehandlung programmiert worden ist. Fällt dieser Test positiv aus, dann rettet der Interpreter die Arbeitsdaten nicht auf den Stapel, sondern in einen eigenen Speicher für TRAP-Daten. Er blockiert dann die Fehlerbehandlung und ruft sich selbst direkt mit dem GOTO-Befehl auf, um die Routine für die Fehlerbehandlung anzuspringen.

9.9 Nutzung der BASIC-ROM-Routinen

9.9.1 Das Sprungmodul

Sicher können Sie sich unter der Überschrift noch nicht viel vorstellen. Aber haben Sie schon einmal die Routinen des Betriebssystems JSRFAR $FF6E und JMPFAR $FF71 benutzt, um ROM-Routinen aufzurufen? Dann wissen Sie sicher, daß es mehr als unangenehm ist, jedesmal alle nötigen Übergabeparameter zu setzen (Siehe Kapitel 7). Diese Arbeit nimmt Ihnen

das Sprungmodul ab. Es erlaubt Ihnen von jeder Stelle im Speicher jede andere anzuspringen, ohne daß Sie sich mit dem Einschalten des ROM oder der Übergabe von Parametern abquälen müssen. Wenn Sie früher das Glockenzeichen $07 ausgeben wollten, so mußten Sie erst mühselig die Zeropageadressen $02 bis $09 setzen, bevor Sie die Ausgabe $FFD2 anspringen konnten. Genau das geht nun einfacher:

```
5000: a9 07      lda #$07
5002: 20 00 17   jsr $1700
5005: 20 d2 ff   jsr $ffd2
5008: 24 0f      bit $0f
500a: ...
```

Sie laden einfach das Glockenzeichen $07 in den Akku und springen das Sprungmodul ab $1700 an. Es erwartet als nächsten Befehl einen JSR mit der Adresse, an die Sie zu springen wünschen. Außerdem ist die Angabe der Bank - in diesem Fall 15 - nötig. Sie brauchen sich auch nicht um irgendwelche Flags zu kümmern. Die Prozessorregister und der Status bleiben ebenso wie der Stackpointer erhalten. Das Programm wird mit dem Befehl fortgesetzt, der auf den BIT-Befehl folgt.
Wollen Sie nicht mit JSR, sondern mit JMP eine andere Routine aufrufen, so sieht das fast genauso aus:

```
5000  a9 07      lda #$07
5002  20 03 17   jsr $1703
5005  4c d2 ff   jmp $ffd2
5008  24 0f      bit $0f
```

Sie sehen, es ist nur ein Sprung nach $1703 und dann ein JMP an die Adresse mit Bank-Angabe erforderlich. Auch hier bleiben alle Register und Flags erhalten.

Diese Möglichkeit, ein Programm zu schreiben, das viele ROM-Routinen benutzt, ist sicher nicht die, die durch überragende Geschwindigkeit überzeugt. Sie hilft jedoch, ein Programm wesentlich übersichtlicher zu gestalten, und macht es außerdem leichter verschiebbar. Natürlich ist das Sprungmodul nicht nur für ROM-Routinen benutzbar. Es erlaubt auch den Ansprung einer anderen RAM-Bank.

Für die besonders Interessierten hier zuerst das Listing des Sprungmoduls im Assembler:

```
1700  4c 40 17   jmp $1740     Einsprung für JSR
1703  4c 63 17   jmp $1763     Einsprung für JMP
1706  85 06      sta $06       Register setzen
1708  86 07      stx $07
170a  84 08      sty $08
170c  08         php           Status setzen
170d  68         pla
170e  85 05      sta $05
1710  ba         tsx
1711  bd 03 01   lda $0103,x   Rücksprungadresse
1714  85 03      sta $03       als Vektor setzen
1716  18         clc
1717  69 05      adc #$05      und hinter Bank-Angabe
1719  9d 03 01   sta $0103,x   setzen
171c  bd 04 01   lda $0104,x
171f  85 04      sta $04
1721  69 00      adc #$00
1723  9d 04 01   sta $0104,x
1726  a0 01      ldy #$01
1728  b1 03      lda ($03),y   Befehlsbyte $20 oder $4C
172a  48         pha           retten
172b  c8         iny
172c  b1 03      lda ($03),y   Zieladresse holen
172e  48         pha           und retten
172f  c8         iny
1730  b1 03      lda ($03),y
1732  aa         tax
1733  c8         iny
1734  c8         iny
1735  b1 03      lda ($03),y   Bank holen
1737  85 02      sta $02       und setzen
1739  86 03      stx $03       Zieladresse setzen
173b  68         pla
173c  85 04      sta $04
173e  68         pla           Befehlsbyte wieder holen
173f  60         rts
1740  20 06 17   jsr $1706     Adresse auswerten
1743  c9 20      cmp #$20      JSR-Befehl ?
1745  f0 01      beq $1748     Ja, Skip
```

```
1747  00        brk         Fehler
1748  ad 00 ff  lda $ff00   MMU-Wert retten
174b  48        pha
174c  29 c0     and #$c0    ROM einblenden
174e  8d 00 ff  sta $ff00
1751  20 6e ff  jsr $ff6e   JSRFAR aufrufen
1754  68        pla
1755  8d 00 ff  sta $ff00   MMU-Wert wieder setzen
1758  a5 05     lda $05     Status holen
175a  48        pha
175b  a5 06     lda $06     Register setzen
175d  a6 07     ldx $07
175f  a4 08     ldy $08
1761  28        plp         und Status setzen
1762  60        rts
1763  20 06 17  jsr $1706   Adresse auswerten
1766  c9 4c     cmp #$4c    JMP-Befehl ?
1768  d0 dd     bne $1747 Nein, Fehler
176a  68        pla         Rücksprungadresse
176b  68        pla         löschen
176c  ad 00 ff  lda $ff00   ROM einblenden
176f  29 c0     and #$c0
1771  8d 00 ff  sta $ff00
1774  4c 71 ff  jmp $ff71   und JMPFAR aufrufen
```

Sie können das Sprungmodul in jeden Speicherbereich legen, der nicht von ROM überdeckt werden kann. Wollen Sie es in beiden Speicherbänken nutzen, so muß es in jeder vorhanden sein. Es ist sehr leicht verschiebbar, da nur vier Adressen geändert werden müssen. Damit Sie nicht soviel Ärger beim Eintippen haben, ist das Sprungmodul hier noch einmal als Speicherauszug aufgelistet. Sie müssen die Zeilen nur mit dem Monitor eingeben und abspeichern.

```
>01700 4c 40 17 4c 63 17 85 06 86 07 84 08 08 68 85 05
>01710 ba bd 03 01 85 03 18 69 05 9d 03 01 bd 04 01 85
>01720 04 69 00 9d 04 01 a0 01 b1 03 48 c8 b1 03 48 c8
>01730 b1 03 aa c8 c8 b1 03 85 02 86 03 68 85 04 68 60
>01740 20 06 17 c9 20 f0 01 00 ad 00 ff 48 29 c0 8d 00
>01750 ff 20 6e ff 68 8d 00 ff a5 05 48 a5 06 a6 07 a4
>01760 08 28 60 20 06 17 c9 4c d0 dd 68 68 ad 00 ff 29
>01770 c0 8d 00 ff 4c 71 ff
```

Das Sprungmodul ist die Grundlage für das folgende Kapitel dieses Buches. Sie sollten es also eintippen. Es sind nicht besonders viele Byte, aber diese wenigen sind sehr nützlich.

9.10 Interessante BASIC-ROM-Routinen

Das BASIC 7.0 Ihres Commodore 128 besitzt zwar keine offiziell anerkannte Sprungvektortabelle wie das Betriebssystem ab $FFxx, es ist aber trotzdem eine vorhanden. Sie liegt ab $AFxx, wie Sie auch im ROM-Listing sehen können, und enthält hauptsächlich Vektoren für die Fließpunktarithmetik. Diese sind natürlich für jede Form von rechenintensiven Programmen sehr nützlich und zum Glück auch einfach zu benutzen. Versuchen wir die einzelnen Routinen etwas näher nach Bedeutung und Bedienung zu beleuchten. Eine Anmerkung sei noch erlaubt; Commodore ändert zwar in den seltensten Fällen etwas freiwillig, aber wenn Commodore etwas macht, so meistens heimlich. Überzeugen Sie sich also besser vorher anhand des ROM-Listings, ob diese Sprungtabelle in Ihrem Commodore 128 überhaupt in diesem Format vorhanden ist!

Doch nun zu den einzelnen Routinen, die in der Sprungtabelle verzeichnet sind: Sind nur sehr wenige Parameter unmittelbar nötig, so sind diese als Eingabeparameter aufgeführt. Sind mehr nötig, sind es im Normalfall so viele, daß eine Aufzählung nicht mehr informativ ist. Auf jeden Fall sollten Sie die einzelnen Routinen auch im ROM-Listing nachschlagen, damit keine Irrtümer entstehen.

Adresse: $AF00
Zweck: FAC#1 in Integerformat bringen

Wenn man diese Routine anspringt, so wird die Fließpunktzahl in FAC#1 in einen Integerwert in $66 und $67 gewandelt, wobei $66 das High-Byte und $67 das Low-Byte ist. Liegt die Realzahl nicht im Integerbereich, so wird eine Fehlermeldung ausgegeben.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF03
Zweck: Integerwert in FAC#1 bringen

Der Integerwert, den Sie im Akku und im Y-Register übergeben, wird in eine Real-Zahl in FAC#1 gewandelt.

Eingabeparameter: .A, .Y
Ausgabeparameter: keine

Adresse: $AF06
Zweck: FAC#1 in ASCII-Format wandeln

Diese Routine wandelt die Realzahl im FAC#1 in einen ASCII-String ab $0100 um. Beendet wird der String durch ein Null-Byte. Übergeben wird im Akku und im Y-Register die Adresse $0100 als Zeiger auf den String.

Eingabeparameter: keine
Ausgabeparameter: .A, .Y

Adresse: $AF09
Zweck: ASCII-Zahlenstring in FAC#1 bringen

Wenn man dieser Routine im Akku die Länge und in ($24) die Adresse eines Zahlenstrings in RAM-Bank 1 übergibt, so wird dieser in eine Realzahl in FAC#1 gewandelt, soweit er als Zahl erkannt werden kann.

Eingabeparameter: .A,($24)
Ausgabeparameter: keine

Adresse: $AF0C
Zweck: FAC#1 in Adreßformat wandeln

Die Realzahl in FAC#1 wird in einen Adreßwert in ($16) gewandelt. Gleichzeitig wird dieser Wert im Y-Register und im Akku übergeben. Bei Bereichsüberschreitung wird eine Fehlermeldung ausgegeben.

Eingabeparameter: keine
Ausgabeparameter: .Y, .A, ($16)

Adresse: $AF0F
Zweck: Adreßwert in FAC#1 bringen

Diese Routine ist nur im Zusammenhang mit $84E5 einfach und sinnvoll zu verwenden! Im Y-Register und im Akku muß der Adreßwert übergeben werden. Dann wird zuerst $84E5 und dann $AF0F angesprungen. Der Adreßwert steht dann als Realzahl in FAC#1.

Eingabeparameter: .Y, .A,Routine $84E5!
Ausgabeparameter: keine

Adresse: $AF12
Zweck: FAC#1 = Konstante - FAC#1

Mit dieser Routine wird vom FAC#1 eine Realkonstante im Speicherformat aus Bank 1 abgezogen. Die Adresse der Konstante wird im Akku und im Y-Register übergeben. Eine Bereichsüberschreitung verursacht eine Fehlermeldung.

Eingabeparameter: .A, .Y
Ausgabeparameter: keine

Adresse: $AF15
Zweck: FAC#1 = FAC#2 - FAC#1

Der FAC#1 wird vom FAC#2 abgezogen, das Ergebnis steht im FAC#1. Bei Bereichsüberschreitung wird eine Fehlermeldung ausgegeben.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF18
Zweck: FAC#1 = Konstante + FAC#1

Mit dieser Routine wird eine Realkonstante im Speicherformat aus Bank 1 auf den FAC#1 aufaddiert. Die Adresse der Konstanten wird im Akku und im Y-Register übergeben. Eine Bereichsüberschreitung verursacht eine Fehlermeldung.

Eingabeparameter: .A, .Y
Ausgabeparameter: keine

Adresse: $AF1B
Zweck: FAC#1 = FAC#2 + FAC#1

Der FAC#2 wird auf den FAC#1 aufaddiert, das Ergebnis steht im FAC#1. Bei Bereichsüberschreitung wird eine Fehlermeldung ausgegeben.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF1E
Zweck: FAC#1 = Konstante * FAC#1

Mit dieser Routine wird eine Realkonstante im Speicherformat aus Bank 1 mit dem FAC#1 multipliziert. Die Adresse der Konstanten wird im Akku und im Y-Register übergeben. Eine Bereichsüberschreitung verursacht eine Fehlermeldung.

Eingabeparameter: .A, .Y
Ausgabeparameter: keine

Adresse: $AF21
Zweck: FAC#1 = FAC#2 * FAC#1

Der FAC#2 wird dem FAC#1 multipliziert, das Ergebnis steht im FAC#1. Bei Bereichsüberschreitung wird eine Fehlermeldung ausgegeben.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF24
Zweck: FAC#1 = Konstante / FAC#1

Mit dieser Routine wird eine Realkonstante im Speicherformat aus Bank 1 durch den FAC#1 geteilt. Die Adresse der Konstanten wird im Akku und im Y-Register übergeben. Eine Bereichsüberschreitung verursacht eine Fehlermeldung.

Eingabeparameter: .A, .Y
Ausgabeparameter: keine

Adresse: $AF27
Zweck: FAC#1 = FAC#2 , FAC#1

Der FAC#2 wird durch den FAC#1 geteilt, das Ergebnis steht im FAC#1. Bei Bereichsüberschreitung wird eine Fehlermeldung ausgegeben.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF2A
Zweck: FAC#1 = LOG(FAC#1)

Diese Routine entspricht der LOG-Funktion des BASIC 7.0. Das Ergebnis steht in FAC#1. Illegale Werte verursachen eine Fehlermeldung.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF2D
Zweck: FAC#1 = INT(FAC#1)

Diese Routine entspricht der INT-Funktion des BASIC 7.0. Das Ergebnis steht in FAC#1. Illegale Werte verursachen eine Fehlermeldung.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF30
Zweck: FAC#1 = SQR(FAC#1)

Diese Routine entspricht der SQR-Funktion des BASIC 7.0. Das Ergebnis steht in FAC#1. Illegale Werte verursachen eine Fehlermeldung.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF33
Zweck: Vorzeichenwechsel FAC#1

Sie können mit dieser Routine ganz einfach das Vorzeichen des FAC#1 wechseln.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF36
Zweck: FAC#1 = FAC#2 ^ Konstante

Der FAC#2 wird mit einer Konstanten im Speicherformat potenziert. Die Adresse der Konstanten wird im Akku und im Y-Register übergeben. Das Ergebnis steht in FAC#1. Illegale Werte verursachen eine Fehlermeldung. Diese Routine ist nur mit Vorsicht zu genießen, da die Konstante aus der aktuellen Speicherkonfiguration geladen wird. Ist an der entsprechenden Stelle ein ROM eingeschaltet, so wird die Konstante auch von dort geladen. Speichern Sie Ihre Konstante einfach in die Bank 0 unter $4000, und geben Sie dem Sprungmodul aus 3.1 als Bankwert 15 an. Sie ersparen sich so Ärger.

Eingabeparameter: .A, .Y
Ausgabeparameter: keine

Adresse: $AF39
Zweck: FAC#1 = FAC#2 ^ FAC#1

Der FAC#2 wird mit dem FAC#1 potenziert. Das Ergebnis steht in FAC#1. Illegale Werte verursachen eine Fehlermeldung. Auch diese Routine verursacht ein Problem. Sie müssen vor dem Aufruf folgenden Befehl ausführen:

```
LDA $63
```

Dann funktioniert sie aber einwandfrei.

Eingabeparameter: .A als Inhalt von $63
Ausgabeparameter: keine

Adresse: $AF3C
Zweck: FAC#1 = EXP(FAC#1)

Diese Routine entspricht der EXP-Funktion des BASIC 7.0. Das Ergebnis steht in FAC#1. Illegale Werte verursachen eine Fehlermeldung.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF3F
Zweck: FAC#1 = COS(FAC#1)

Diese Routine entspricht der COS-Funktion des BASIC 7.0. Das Ergebnis steht in FAC#1. Illegale Werte verursachen eine Fehlermeldung.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF42
Zweck: FAC#1 = SIN(FAC#1)

Diese Routine entspricht der SIN-Funktion des BASIC 7.0. Das Ergebnis steht in FAC#1. Illegale Werte verursachen eine Fehlermeldung.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF45
Zweck: FAC#1 = TAN(FAC#1)

Diese Routine entspricht der TAN-Funktion des BASIC 7.0. Das Ergebnis steht in FAC#1. Illegale Werte verursachen eine Fehlermeldung.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF48
Zweck: FAC#1 = ATN(FAC#1)

Diese Routine entspricht der ATN-Funktion des BASIC 7.0. Das Ergebnis steht in FAC#1. Illegale Werte verursachen eine Fehlermeldung.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF4B
Zweck: FAC#1 runden

Der FAC#1 wird gerundet.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF4E
Zweck: FAC#1 = ABS(FAC#1)

Diese Routine entspricht der ABS-Funktion des BASIC 7.0. Das Ergebnis steht in FAC#1. Illegale Werte verursachen eine Fehlermeldung.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF51
Zweck: Vorzeichen von FAC#1 holen

Nach Ansprung dieser Routine steht im Akku das Vorzeichen vom FAC#1. Der Wert $01 bedeutet größer Null, $FF kleiner Null und $00 bedeutet, daß der FAC#1 gleich Null ist.

Eingabeparameter: keine
Ausgabeparameter: .A

Adresse: $AF54
Zweck: Vergleich Konstante mit FAC#1

Diese Routine vergleicht eine Realkonstante im Speicherformat mit dem FAC#1. Die Adresse der Konstante wird im Akku und im Y-Register übergeben. Das Vergleichsergebnis wird im Akku übergeben. Es entspricht dem der Routine $AF51, der Vergleichswert ist aber nicht Null, sondern die Konstante. Hier tritt dasselbe Problem wie bei $AF36 auf. Die Konstante wird aus der aktuellen Speicherkonfiguration geladen, also auch aus dem ROM, falls dies eingeschaltet ist. Legen Sie Ihre Konstante also ins RAM unter $4000, um Probleme zu vermeiden.

Eingabeparameter: .A, .Y
Ausgabeparameter: .A

Adresse: $AF57
Zweck: FAC#1 = RND(FAC#1)

Diese Routine entspricht der RND-Funktion des BASIC 7.0, das Ergebnis steht im FAC#1. Bevor diese Routine aufgerufen wird, muß folgender Befehl ausgeführt werden, um die Flags zu setzen:

```
LDA $63
```

Dann gibt es aber ebenso wie bei $AF39 keine Probleme mehr.

Eingabeparameter: .A als Inhalt von $63
Ausgabeparameter: keine

Adresse: $AF5A
Zweck: FAC#2 = Konstante aus Bank 1

Eine Realkonstante aus Bank 1 wird in den FAC#2 geladen. Die Adresse der Konstanten wird im Akku und im Y-Register

übergeben. Im Akku bekommen Sie den Exponenten des FAC#1 als Wert zurück.

Eingabeparameter: .A, .Y
Ausgabeparameter: .A

Adresse: $AF5D
Zweck: FAC#2 = Konstante

Eine Realkonstante wird in den FAC#2 geladen. Die Adresse der Konstanten wird im Akku und im Y-Register übergeben. Zurückgegeben wird im Akku der Exponent des FAC#1. Auch hier muß man wieder die aktuelle Speicherkonfiguration beachten und seine Konstante in einen mit Sicherheit freien RAM-Bereich legen.

Eingabeparameter: .A, .Y
Ausgabeparameter: .A

Adresse: $AF60
Zweck: FAC#1 = Konstante aus Bank 1

Diese Routine hat für den FAC#1 dieselbe Funktion wie $AF5A. Es wird aber kein Exponentwert zurückgegeben.

Eingabeparameter: .A, .Y
Ausgabeparameter: keine

Adresse: $AF63
Zweck: FAC#1 = Konstante

Diese Routine hat für den FAC#1 dieselbe Funktion wie $AF5D. Der Exponent des FAC#1 wird im Akku zurückgegeben.

Eingabeparameter: .A, .Y
Ausgabeparameter: .A

Adresse: $AF66
Zweck: FAC#1 in den Speicher mit Rundung

Der Inhalt des FAC#1 wird als Konstante im Speicherformat in der aktuellen Speicherbank abgelegt. Die Speicheradresse wird im X-Register und im Y-Register übergeben.

Eingabeparameter: .X, .Y
Ausgabeparameter: keine

Adresse: $AF69
Zweck: FAC#1 = FAC#2

Der Inhalt des FAC#2 wird in den FAC#1 kopiert.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF6C
Zweck: FAC#2 = FAC#1 mit Rundung

Der FAC#1 wird gerundet und in den FAC#2 kopiert.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF6F
Zweck: Adresse der Hierarchiewerttabelle

Dies ist keine ROM-Routine! Hier ist nur die Adresse der Hierarchiewerttabelle der Operatoren für die Auswertung von Ausdrücken FRMEVL $77EF abgelegt.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF72
Zweck: Linie von Start- zu Ziel zeichnen

Diese Routine entspricht in etwa dem DRAW-Befehl des BASIC 7.0. Sie müssen jedoch selber die Farbquelle $83, die Koordinatenwerte ab $1131 und den Rest der Grafik verwalten.

Eingabeparameter: Koordinatenvorgaben
Ausgabeparameter: keine

Adresse: $AF75
Zweck: Koordinaten setzen

Diese Routine setzt auf dem Schirm einen Punkt an den aktuellen Grafikkoordinaten ab $1131. Sie berücksichtigt dabei auch die Angabe für doppelte Breite des BASIC-Befehls WIDTH und die des BOX-Befehls, die in $116B und in $116C gespeichert sind. Sie müssen wie bei $AF72 daran denken, daß Sie die gesamte Grafik selbst verwalten müssen!

Eingabeparameter: Koordinatenvorgaben
Ausgabeparameter: keine

Adresse: $AF78
Zweck: Nächsten Kreisbogenpunkt berechnen

Hier müssen Sie relativ umfangreiche Vorbereitungen treffen, um diese Routine überhaupt nutzen zu können. Sie sollten den CIRCLE-Befehl im ROM-Listing ausgiebig studieren.

Eingabeparameter: Koordinatenvorgaben
Ausgabeparameter: keine

Adresse: $AF7B
Zweck: RUN-Befehl

Diese Routine ist der Direkteinsprung in den RUN-Befehl, wie ihn der Interpreter machen würde. Sie müssen also den PC und die restlichen Flags des Interpreters richtig verwalten, da sonst

eine Fehlermeldung ausgegeben wird. Vergessen Sie nicht, vor dem Aufruf CHRGOT $0386 anzuspringen. Vom RUN-Befehl wird direkt in die Interpreterschleife gesprungen, der Computer kehrt also nicht in Ihr Programm zurück.

Eingabeparameter: CHRGOT,Flags
Ausgabeparameter: keine

Adresse: $AF7E
Zweck: PC aus Programmstart, CLR

Diese Routine setzt den PC auf den Programmstart und führt dann den CLR-Befehl aus. Damit wird auch der Prozessorstack zurückgesetzt. Sie können deshalb diese Routine ausschließlich direkt anspringen, da bis auf die aktuelle Rücksprungadresse der Stack gelöscht wird. Hier ist Ihnen das Sprungmodul nicht von Nutzen.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF81
Zweck: CLR-Befehl

Diese Routine entspricht dem CLR-Befehl des BASIC 7.0. Sie müssen wie bei $AF7B vorher alle Flags setzen, die der Interpreter benötigt. Außerdem hat diese Routine dieselben Schwierigkeiten beim Aufruf wie $AF7E. Auch hier können Sie also das Sprungmodul nicht verwenden.

Eingabeparameter: CHRGOT,Flags
Ausgabeparameter: keine

Adresse: $AF84
Zweck: NEW-Befehl

Diese Routine entspricht dem NEW-Befehl des BASIC 7.0. Die Handhabung ist dieselbe, wie bei $AF81

Eingabeparameter: CHRGOT,Flags
Ausgabeparameter: keine

Adresse: $AF87
Zweck: Zeilenverkettung korrigieren

Die Verkettungsadressen der Programmzeilen werden neu berechnet. Der Interpreter benutzt dies zum Beispiel nach dem Löschen von Zeilen im Programm. Für den GOTO-Befehl ist ein richtig verkettetes Programm wichtig. Rufen Sie also diese Routine auf, wenn Ihr Programm das BASIC-Programm verändert hat.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF8A
Zweck: Umwandlung in Interpretercode

Der Text, auf den der PC in $3D zeigt, wird bis zum Zeilenende, das durch ein Null-Byte markiert ist, in Interpretercode umgewandelt. Der Klartext der Befehle wird also durch Tokens ersetzt. Normalerweise wird die Eingabezeile ab $0200 abgelegt und der PC darauf gerichtet. Sie könnten diese Routine verwenden, um Eingaben Ihres Programms in Interpretercode zu verwandeln und um daraus dann ein BASIC-Programm zu machen.

Eingabeparameter: PC
Ausgabeparameter: keine

Adresse: $AF8D
Zweck: Zeile suchen

Die Programmzeile wird gesucht, deren Zeilennummer in ($16) abgelegt ist. Wird die Zeile gefunden, so ist das C-Flag des Prozessorstatus gesetzt. Die Adresse der Programmzeile ist dann in ($61) abgelegt.

Eingabeparameter: ($16)
Ausgabeparameter: C-Flag, ($61)

Adresse: $AF90
Zweck: Sprung an Interpreterschleifenstart

Die Interpreterschleife wird angesprungen. Alle Flags des Interpreters und der PC müssen richtig vorbesetzt sein. Der Prozessor kehrt nicht wieder in Ihr Programm zurück.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF93
Zweck: Nächstes Ausdruckelement holen

Sollten Sie eigene Auswertungen in Ihrem Programm vornehmen, so können Sie dieses Unterprogramm von FRMEVL $77EF verwenden. Es wertet die nächste Zahl, den nächsten String, die nächste Funktion etc. aus, auf die der PC zeigt. Das Ergebnis steht immer im FAC#1. Hier ist ein Studium des ROM-Listings auf jeden Fall nötig.

Eingabeparameter: keine
Ausgabeparameter: FAC#1

Adresse: $AF96
Zweck: FRMEVL Ausdruck auswerten

Ein beliebiger Ausdruck aus dem Programmtext auf den der PC zeigt wird ausgewertet. Die Flags $0F und $10, die den Typ des Ergebnisses angeben, werden entsprechend gesetzt. Das Ergebnis, das natürlich auch ein Zeiger auf einen String sein kann, steht im FAC#1.

Eingabeparameter: keine
Ausgabeparameter: FAC#1

Adresse: $AF99
Zweck: RUN-Direkteinsprung

Dies ist ein Direkteinsprung in den RUN-Befehl des BASIC 7.0 und wirkt wie ein RUN ohne Angabe einer Zeilennummer. Da direkt in die Interpreterschleife eingesprungen wird, kehrt der Prozessor nicht in Ihr Programm zurück.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF9C
Zweck: Programm-Modus setzen

Der Programm-Modus wird eingeschaltet. Dabei werden der AUTO-Befehl, die BASIC-Interrupts und der Auto-Insert-Modus des Bildschirmeditors ausgeschaltet. Das Programmflag des Interpreters wird gesetzt und das Betriebssystem unterdrückt ab jetzt seine Meldungen.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AF9F
Zweck: Zeilennummer lesen

Eine Zeilennummer wird aus dem Programmtext eingelesen,auf den der PC zeigt und in ($16) abgelegt. Die Anzahl der eingelesenen Ziffern wird in $0A gespeichert. Vor dem Aufruf dieser Routine muß CHRGOT aufgerufen werden, um alle Flags zu setzen.

Eingabeparameter: CHRGOT,Flags
Ausgabeparameter: ($16),$0A

Adresse: $AFA2
Zweck: Garbage-Collection

Der Interpreter führt eine "Müllsammlung" durch, das heißt er bereinigt seinen Stringspeicher und löscht alle ungültigen Strings. Bei Programmen, die sehr viel mit Strings arbeiten, ist das hin und wieder nötig, damit in der Variablenbank 1 wieder freier Speicherplatz vorhanden ist.

Eingabeparameter: keine
Ausgabeparameter: keine

Adresse: $AFA5
Zweck: Direkteinsprung in Eingabeschleife

Wenn Sie im X-Register und im Y-Register die Adresse minus 1 einer eigenen Eingabezeile in Bank 0 angeben und diese Routine anspringen, so wird diese Eingabe so ausgewertet, wie der Interpreter eine Eingabe im Direktmodus auswerten würde. Der Prozessor kehrt aber nicht in Ihr Programm zurück. Der Interpreter läuft dann nämlich wieder in seiner Endlosschleife.

Eingabeparameter: .X, .Y
Ausgabeparameter: keine

Sie sehen, allein ein Kurzkommentar zu den ROM-Routinen der Sprungtabelle verbraucht einige Seiten dieses Buches. Es ist sicher verständlich, daß nicht die Handhabung jeder einzelnen Routine dokumentiert werden kann. Aber da Sie auch die meisten anderen Routinen, die im ROM-Listing und in der Speicherbelegung aufgeführt worden sind, benutzen können, werden hier noch einige Tips gegeben:

- Das BASIC 7.0 ist sehr flexibel. Die einzelnen Routinen erfordern deshalb oft viele Parameter. Haben Sie alle beachtet?

- Einige Routinen verändern den Stackpointer. Stürzt das Sprungmodul oder Ihr eigener Aufruf deshalb ab?

- Viele ROM-Routinen schalten zwischen den Speicherbanken um. Haben Sie die richtige Speicherkonfiguration gewählt?

- BASIC-Befehle werden immer nach einem Aufruf von CHRGET $0380 angesprungen. Haben Sie das vergessen?

- Der Interpreter "kramt" ausgiebig im Speicher. Liegt Ihr Programm auch wirklich in einem geschützten Bereich?

Manche dieser Anmerkungen mögen Ihnen dumm erscheinen, sie sind aber alle aufgrund von eigenen Erfahrungen entstanden. Der Commodore 128 ist komplexer als man glaubt. Nachdem alle eventuellen Schwierigkeiten beseitigt sind, hier nun das sorgsam dokumentierte BASIC-ROM-Listing:

```
******************************* Sprungtabelle BASIC-Start

4000   4C 23 40   JMP $4023    BASIC-Kaltstart

4003   4C 09 40   JMP $4009    BASIC-Warmstart

4006   4C 4D A8   JMP $A84D    BASIC-IRQ-Routine

******************************* BASIC-Warmstart

4009   20 CC FF   JSR $FFCC    I/O-Kanäle rücksetzen
400C   20 7A 41   JSR $417A    MMU programmieren
400F   20 8D 41   JSR $418D    Sprite-Daten setzen
4012   20 12 41   JSR $4112    Sound-Daten setzen
4015   20 38 52   JSR $5238    Stack rücksetzen
4018   A9 00      LDA #$00     Momentanes I/O-Gerät = Tastatur
401A   85 15      STA $15
401C   58         CLI
401D   4C 37 4D   JMP $4D37    'READY.'

******************************* Füllbytes

4020   00 FF FF

******************************* BASIC-Kaltstart

4023   20 7A 41   JSR $417A    MMU programmieren
4026   20 51 42   JSR $4251    BASIC-Vektoren setzen
4029   20 45 40   JSR $4045    BASIC-Zeiger setzen
402C   20 9B 41   JSR $419B    Copyright ausgeben
402F   AD 04 0A   LDA $0A04    BASIC-IRQ Aufruf erlauben
4032   09 01      ORA #$01
4034   8D 04 0A   STA $0A04
4037   A2 03      LDX #$03     BASIC-Startvektor auf Warmstart
4039   8E 00 0A   STX $0A00    $4003
403C   A2 FB      LDX #$FB     Stapel rücksetzen
403E   9A         TXS
403F   20 56 FF   JSR $FF56    Modul- und BOOT-Test
4042   4C 1C 40   JMP $401C    'READY.'

******************************* BASIC-Zeiger setzen

4045   A9 4C      LDA #$4C     JMP für Funktionsauswertung
4047   85 56      STA $56      setzen
4049   8D 18 12   STA $1218    JMP für USR-Befehl setzen
404C   A9 28      LDA #$28     USR-Adresse auf $7D28 setzen
404E   A0 7D      LDY #$7D     'ILLEGAL QUANTITY'
4050   8D 19 12   STA $1219
```

```
4053  8C 1A 12  STY $121A
4056  A9 9F     LDA #$9F     Vektor Fließpunkt zu Integer
4058  A0 84     LDY #$84     auf $849F setzen
405A  8D 7A 11  STA $117A
405D  8C 7B 11  STY $117B
4060  A9 3C     LDA #$3C     Vektor Integer zu Fließpunkt
4062  A0 79     LDY #$79     auf $793C setzen
4064  8D 7C 11  STA $117C
4067  8C 7D 11  STY $117D
406A  A2 55     LDX #$55     BASIC-Routinen und Konstanten
406C  BD 78 42  LDA $4278,X  in Zero-Page
406F  9D 7F 03  STA $037F,X
4072  CA        DEX
4073  D0 F7     BNE $406C
4075  8E DF 03  STX $03DF    Überlauf FAC#1 löschen
4078  86 15     STX $15      Aktives I/O-Gerät = Tastatur
407A  86 1A     STX $1A      Stringstack löschen
407C  8E 6F 11  STX $116F    TRACE ausschalten
407F  8E 00 1C  STX $1C00    Programmstartbyte auf 0
4082  86 76     STX $76      AUTO ausschalten
4084  86 74     STX $74
4086  86 75     STX $75
4088  8E 6B 11  STX $116B    Skalierung = 320 * 200
408B  8E 6A 11  STX $116A    Doppelte Breite löschen
408E  8E 6C 11  STX $116C    Ausfüllflag löschen
4091  8E 1B 12  STX $121B    RND-Exponent auf 0
4094  8E 1C 01  STX $011C    DOS-Gerätenummer löschen
4097  8E 76 12  STX $1276    Interruptspeicher löschen
409A  8E 77 12  STX $1277
409D  8E 78 12  STX $1278
40A0  8E 7F 12  STX $127F
40A3  A0 58     LDY #$58
40A5  99 7E 11  STA $117E,Y  Spriteparameter löschen
40A8  88        DEY
40A9  10 FA     BPL $40A5
40AB  E8        INX
40AC  8E FD 01  STX $01FD
40AF  8E FC 01  STX $01FC
40B2  A2 0F     LDX #$0F     BANK für SYS,POKE,PEEK auf
40B4  8E D5 03  STX $03D5    15 setzen
40B7  A2 0D     LDX #$0D     Vordergrundfarbe = Hellgrün
40B9  86 86     STX $86
40BB  A2 01     LDX #$01     Multicolorfarbe 1 = Weiß
40BD  86 84     STX $84
40BF  A2 02     LDX #$02     Multicolorfarbe 2 = Rot
40C1  86 85     STX $85
40C3  20 5C 6A  JSR $6A5C    Gepackte Farbwerte setzen
40C6  A2 1B     LDX #$1B     Stringstack rücksetzen
```

```
40C8   86 18      STX $18
40CA   A2 01      LDX #$01     Speicherstart für Programm in Bank 0
40CC   A0 1C      LDY #$1C     auf $1C01 setzen
40CE   86 2D      STX $2D
40D0   84 2E      STY $2E
40D2   A9 00      LDA #$00     Speicherstart für Variablen in Bank 1
40D4   A0 04      LDY #$04     auf $0400 setzen
40D6   85 2F      STA $2F
40D8   84 30      STY $30
40DA   A9 00      LDA #$00     Speicherende für Programm in Bank 0
40DC   A0 FF      LDY #$FF     auf $FF00 setzen
40DE   8D 12 12   STA $1212
40E1   8C 13 12   STY $1213
40E4   A9 00      LDA #$00     Speicherende für Variablen in Bank 1
40E6   A0 FF      LDY #$FF     auf $FF00 setzen
40E8   85 39      STA $39
40EA   84 3A      STY $3A
40EC   A2 3F      LDX #$3F     Spritepointer setzen
40EE   A0 07      LDY #$07
40F0   8A         TXA
40F1   99 F8 07   STA $07F8,Y
40F4   CA         DEX
40F5   88         DEY
40F6   10 F8      BPL $40F0
40F8   A9 00      LDA #$00     Alle VIC-Daten löschen
40FA   A2 6C      LDX #$6C
40FC   9D 7E 11   STA $117E,X
40FF   CA         DEX
4100   10 FA      BPL $40FC
4102   20 12 41   JSR $4112    Sound-Daten setzen
4105   A9 D0      LDA #$D0     Zeiger auf Zeichengenerator setzen
4107   8D EC 11   STA $11EC    (für CHAR-Befehl)
410A   A9 D8      LDA #$D8
410C   8D EB 11   STA $11EB
410F   4C D9 51   JMP $51D9    NEW-Befehl aufrufen

****************************** Sound-Daten vorbesetzen

4112   A9 20      LDA #$20     Tempo vorbesetzen
4114   A0 01      LDY #$01
4116   8D 29 12   STA $1229
4119   8C 2A 12   STY $122A
411C   A9 04      LDA #$04     Oktave auf 4
411E   8D 2B 12   STA $122B
4121   A9 10      LDA #$10     Tempo auf 16
4123   8D 22 12   STA $1222
4126   A9 00      LDA #$00     Alle 3 Stimmen löschen
4128   8D 04 D4   STA $D404
```

```
412B   8D 0B D4   STA $D40B
412E   8D 12 D4   STA $D412
4131   8D FD 12   STA $12FD     Interruptspeicher löschen
4134   A9 0F      LDA #$0F      Lautstärken setzen
4136   8D 74 12   STA $1274
4139   8D 75 12   STA $1275
413C   8D 18 D4   STA $D418     Maximallautstärke
413F   A0 1D      LDY #$1D
4141   B9 11 70   LDA $7011,Y   Klangwerte vorbesetzen
4144   99 3F 12   STA $123F,Y
4147   88         DEY
4148   10 F7      BPL $4141
414A   A2 09      LDX #$09
414C   BD 2F 70   LDA $702F,X
414F   9D 67 12   STA $1267,X   Pulswert High setzen
4152   CA         DEX
4153   10 F7      BPL $414C
4155   8E 85 12   STX $1285     Soundspeicher löschen
4158   8E 86 12   STX $1286
415B   8E 87 12   STX $1287
415E   8E 24 12   STX $1224     Playspeicher löschen
4161   8E 26 12   STX $1226
4164   8E 28 12   STX $1228
4167   A0 02      LDY #$02      Stimmenzähler auf Stimme 3 (0 bis 2)
4169   8C 2F 12   STY $122F
416C   A2 00      LDX #$00
416E   20 B2 6E   JSR $6EB2     Stimme vorbesetzen
4171   CE 2F 12   DEC $122F
4174   10 F6      BPL $416C
4176   EE 2F 12   INC $122F     Stimmennummer auf 0
4179   60         RTS

******************************** MMU programmieren

417A   20 45 A8   JSR $A845     ROMs einschalten
417D   A2 03      LDX #$03
417F   BD 89 41   LDA $4189,X   MMU programmieren
4182   9D 01 D5   STA $D501,X
4185   CA         DEX
4186   10 F7      BPL $417F
4188   60         RTS

******************************** Sprite-Daten setzen

418D   A9 00      LDA #$00      Geschwindingkeit 0
418F   A0 07      LDY #$07      Sprites 0 bis 7 verarbeiten
4191   BE D9 6D   LDX $6DD9,Y   Offset auf Tabelle laden
4194   9D 7E 11   STA $117E,X   Geschwindigkeit auf 0
```

```
4197   88         DEY
4198   10 F7      BPL $4191
419A   60         RTS

****************************** Copyright ausgeben

419B   A0 00      LDY #$00
419D   B9 BB 41   LDA $41BB,Y
41A0   C9 40      CMP #$40      Zentrieren ?
41A2   D0 0E      BNE $41B2     Nein, dann ausgeben
41A4   24 D7      BIT $D7       80-Zeichen-Modus ?
41A6   10 0D      BPL $41B5     Nein, dann nächstes Zeichen
41A8   A2 13      LDX #$13      19 mal Space auf 80-Zeichen-Schirm
41AA   A9 20      LDA #$20
41AC   20 69 92   JSR $9269     Zeichen ausgeben
41AF   CA         DEX
41B0   D0 F8      BNE $41AA
41B2   20 69 92   JSR $9269     Textzeichen ausgeben
41B5   C8         INY
41B6   C0 96      CPY #$96      Alles ausgegeben ?
41B8   D0 E3      BNE $419D     Nein, weitermachen
41BA   60         RTS

****************************** Copyright-Text

41BB   93 0D 40 20 43 4F 4D 4D (CLR) (CR) 'COMMODORE BASIC'
41C3   4F 44 4F 52 45 20 42 41       'v7.0 122365 BYTES FREE'
41CB   53 49 43 20 56 37 2E 30 (CR) '(C)1985 COMMODORE ELECTRONICS,'
41D3   20 31 32 32 33 36 35 20       'ltd.' (CR)
41DB   42 59 54 45 53 20 46 52       '(C)1977 MICROSOFT CORP.' (CR)
41E3   45 45 0D 40 20 20 20 28       'ALL RIGHTS RESERVED' (CR)
41EB   43 29 31 39 38 35 20 43
41F3   4F 4D 4D 4F 44 4F 52 45
41FB   20 45 4C 45 43 54 52 4F
4203   4E 49 43 53 2C 20 4C 54
420B   44 2E 0D 40 20 20 20 20
4213   20 20 20 20 20 28 43 29
421B   31 39 37 37 20 4D 49 43
4223   52 4F 53 4F 46 54 20 43
422B   4F 52 50 2E 0D 40 20 20
4233   20 20 20 20 20 20 20 20
423B   20 41 4C 4C 20 52 49 47
4243   48 54 53 20 52 45 53 45
424B   52 56 45 44 0D 00

****************************** BASIC-Vektoren setzen

4251   A2 11      LDX #$11
```

```
4253  BD 67 42   LDA $4267,X
4256  9D 00 03   STA $0300,X  Vektoren setzen
4259  CA         DEX
425A  10 F7      BPL $4253
425C  A9 78      LDA #$78     Vektor für Zusatzfunktionen
425E  8D FC 02   STA $02FC    auf $4C78 setzen
4261  A9 4C      LDA #$4C
4263  8D FD 02   STA $02FD
4266  60         RTS

****************************** BASIC-Vektortabelle

4267  3F 4D C6 4D 0D 43 51 51
426F  A2 4A DA 78 21 43 CD 51
4277  A9 4B

****************************** ROM-Kopie der Zero-Page-Routinen

4279  E6 3D      INC $3D      Kopie der CHRGET-Routine
427B  D0 02      BNE $427F
427D  E6 3E      INC $3E
427F  8D 01 FF   STA $FF01    RAM-Bank 0 ein
4282  A0 00      LDY #$00
4284  B1 3D      LDA ($3D),Y
4286  8D 03 FF   STA $FF03    ROM wieder ein
4289  C9 3A      CMP #$3A     >= ':' ?
428B  B0 0A      BCS $4297    Ja, SEC, Ende
428D  C9 20      CMP #$20     Space ?
428F  F0 E8      BEQ $4279    Ja, überlesen
4291  38         SEC          Zahlentest
4292  E9 30      SBC #$30     Bei Zahl CLC
4294  38         SEC
4295  E9 D0      SBC #$D0
4297  60         RTS

4298  8D A6 03   STA $03A6    Load aus RAM-Bank 0
429B  8D 01 FF   STA $FF01    Read aus Bank 0 setzen
429E  B1 00      LDA ($00),Y  $00=$03A6 !
42A0  8D 03 FF   STA $FF03    ROMs wieder ein
42A3  60         RTS

42A4  8D B2 03   STA $03B2    Load aus RAM-Bank 1
42A7  8D 02 FF   STA $FF02    Read aus Bank 1 setzen
42AA  B1 00      LDA ($00),Y  $00=$03B2 !
42AC  8D 04 FF   STA $FF04    ROMs wieder ein
42AF  60         RTS

42B0  8D 02 FF   STA $FF02    Load aus RAM-Bank 1
```

```
42B3   B1 24      LDA ($24),Y
42B5   8D 04 FF   STA $FF04     ROMs wieder ein
42B8   60         RTS

42B9   8D 01 FF   STA $FF01     Load aus RAM-Bank 0
42BC   B1 26      LDA ($26),Y
42BE   8D 03 FF   STA $FF03     ROMs wieder ein
42C1   60         RTS

42C2   8D 01 FF   STA $FF01     Load aus RAM-Bank 0
42C5   B1 3D      LDA ($3D),Y
42C7   8D 03 FF   STA $FF03     ROMs wieder ein
42CA   60         RTS

****************************** Füllbytes

42CB   00 00 00

****************************** Verschiedene Bank-Loads

42CE   A9 50      LDA #$50      Bank 1 LDA ($50),Y
42D0   4C AB 03   JMP $03AB

42D3   A9 3F      LDA #$3F      Bank 1 LDA ($3F),Y
42D5   4C AB 03   JMP $03AB

42D8   A9 52      LDA #$52      Bank 1 LDA ($52),Y
42DA   4C AB 03   JMP $03AB

42DD   A9 5C      LDA #$5C      Bank 0 LDA ($5C),Y
42DF   4C 9F 03   JMP $039F

42E2   A9 5C      LDA #$5C      Bank 1 LDA ($5C),Y
42E4   4C AB 03   JMP $03AB

42E7   A9 66      LDA #$66      Bank 1 LDA ($66),Y
42E9   4C AB 03   JMP $03AB

42EC   A9 61      LDA #$61      Bank 0 LDA ($61),Y
42EE   4C 9F 03   JMP $039F

42F1   A9 70      LDA #$70      Bank 0 LDA ($70),Y
42F3   4C 9F 03   JMP $039F

42F6   A9 70      LDA #$70      Bank 1 LDA ($70),Y
42F8   4C AB 03   JMP $03AB

42FB   A9 50      LDA #$50      Bank 1 LDA ($50),Y
```

```
42FD   4C AB 03   JMP $03AB

4300   A9 61      LDA #$61      Bank 1 LDA ($61),Y
4302   4C AB 03   JMP $03AB

4305   A9 24      LDA #$24      Bank 0 LDA ($24),Y
4307   4C 9F 03   JMP $039F

****************************** Umwandlung in Interpretercode

430A   6C 04 03   JMP ($0304)   $430D Umwandlung in Interpretercode
430D   A5 3D      LDA $3D       PC retten
430F   48         PHA
4310   A5 3E      LDA $3E
4312   48         PHA
4313   20 86 03   JSR $0386     CHRGOT
4316   4C 1C 43   JMP $431C

4319   20 80 03   JSR $0380     CHRGET
431C   90 FB      BCC $4319     Wenn Zahl,dann überlesen
431E   6C 0C 03   JMP ($030C)   $4321 ESC-Umwandlung
4321   B0 03      BCS $4326     Wenn nicht erkannt, normal testen
4323   4C B2 43   JMP $43B2     Token als Sondertoken eintragen

4326   C9 00      CMP #$00      Zeilenende ?
4328   F0 77      BEQ $43A1     Ja, Ende
432A   C9 3A      CMP #$3A      Doppelpunkt ?
432C   F0 EB      BEQ $4319     Ja, überlesen
432E   C9 3F      CMP #$3F      Fragezeichen ?
4330   D0 04      BNE $4336     Nein, Skip
4332   A9 99      LDA #$99      '?' durch PRINT ersetzen
4334   D0 50      BNE $4386     und eintragen

4336   C9 80      CMP #$80      Token ?
4338   90 0B      BCC $4345     Nein,Skip
433A   C9 FF      CMP #$FF      Token für PI ?
433C   F0 DB      BEQ $4319     Ja, überlesen
433E   A0 01      LDY #$01      Tokencode löschen
4340   20 CC 43   JSR $43CC
4343   F0 CE      BEQ $4313     Unbedingter Sprung

4345   C9 22      CMP #$22      Anführungszeichen ?
4347   D0 0D      BNE $4356     Nein, Skip
4349   20 80 03   JSR $0380     Nächstes Zeichen holen
434C   C9 00      CMP #$00      Zeilenende ?
434E   F0 51      BEQ $43A1     Ja, Ende
4350   C9 22      CMP #$22      Zweites '"' ?
4352   F0 C5      BEQ $4319     Ja, zum Schleifenstart
```

```
4354   D0 F3      BNE $4349    Nein, weiterüberlesen

4356   A9 46      LDA #$46     Befehl ab $4609 ?
4358   A0 09      LDY #$09
435A   20 E2 43   JSR $43E2
435D   90 06      BCC $4365    Nein, Skip
435F   A9 81      LDA #$81     Tokenoffset
4361   A2 00      LDX #$00     Sondertokenwert 1
4363   F0 4B      BEQ $43B0    Unbedingter Sprung

4365   A9 46      LDA #$46     Befehl ab $49C9 ?
4367   A0 C9      LDY #$C9
4369   20 E2 43   JSR $43E2
436C   90 06      BCC $4374    Nein, Skip
436E   A9 81      LDA #$81     Tokenoffset
4370   A2 FF      LDX #$FF     Sondertokenwert 2
4372   D0 3C      BNE $43B0    Unbedingter Sprung

4374   A9 44      LDA #$44     Befehl ab $4417 ?
4376   A0 17      LDY #$17
4378   20 E2 43   JSR $43E2
437B   90 9C      BCC $4319    Nein, überlesen
437D   C0 00      CPY #$00     Befehlslänge ist 1 Zeichen ?
437F   F0 03      BEQ $4384    Ja, Skip
4381   20 CC 43   JSR $43CC    Falls nötig, Zeichen löschen
4384   A5 0D      LDA $0D      Tokennummer
4386   A0 00      LDY #$00
4388   91 3D      STA ($3D),Y  in Textzeile
438A   C9 8F      CMP #$8F     REM ?
438C   F0 0D      BEQ $439B    Ja, Skip
438E   C9 83      CMP #$83     DATA ?
4390   D0 87      BNE $4319    Nein, weitertesten
4392   20 80 03   JSR $0380    CHRGET
4395   20 8F 52   JSR $528F    Nächstes Trennzeichen aufsuchen
4398   4C 13 43   JMP $4313    Weitertesten

439B   20 80 03   JSR $0380    CHRGET
439E   20 9D 52   JSR $529D    REM-Zeile überlesen
43A1   A6 3D      LDX $3D
43A3   68         PLA          PC holen
43A4   85 3E      STA $3E
43A6   68         PLA
43A7   85 3D      STA $3D
43A9   38         SEC
43AA   8A         TXA
43AB   E5 3D      SBC $3D
43AD   A8         TAY
43AE   C8         INY          (Y) = Zeilenlänge
```

```
43AF   60         RTS

43B0   65 0D      ADC $0D        Tokennummer korrigieren
43B2   48         PHA
43B3   88         DEY
43B4   20 CC 43   JSR $43CC      Befehlszeichen löschen
43B7   A9 FE      LDA #$FE       Sondertokenwert 1 laden
43B9   E8         INX            Sondertoken 1 gefragt ?
43BA   D0 02      BNE $43BE      Ja, Skip
43BC   A9 CE      LDA #$CE       Sondertokenwert 2 laden
43BE   A0 00      LDY #$00       Sondertoken in Textzeile
43C0   91 3D      STA ($3D),Y
43C2   C8         INY
43C3   68         PLA            Tokenwert holen
43C4   91 3D      STA ($3D),Y    und in Befehlszeile
43C6   20 80 03   JSR $0380      CHRGET
43C9   4C 19 43   JMP $4319      Weitertesten

****************************** (Y) Zeichen aus Zeile löschen

43CC   18         CLC
43CD   98         TYA            Länge auf PC aufaddieren
43CE   65 3D      ADC $3D
43D0   85 24      STA $24
43D2   A5 3E      LDA $3E
43D4   69 00      ADC #$00
43D6   85 25      STA $25
43D8   A0 FF      LDY #$FF       Offset auf Start
43DA   C8         INY
43DB   B1 24      LDA ($24),Y    Zeichen umkopieren
43DD   91 3D      STA ($3D),Y
43DF   D0 F9      BNE $43DA      bis Zeilenende
43E1   60         RTS

****************************** Befehl erkennen

43E2   85 25      STA $25        Zeiger auf Befehlstabelle setzen
43E4   84 24      STY $24
43E6   A0 00      LDY #$00       Befehlsnummer auf 0
43E8   84 0D      STY $0D
43EA   88         DEY
43EB   C8         INY
43EC   B1 3D      LDA ($3D),Y    Textzeichen
43EE   38         SEC            gleich
43EF   F1 24      SBC ($24),Y    Zeichen aus Tabelle ?
43F1   F0 F8      BEQ $43EB      Ja, weitervergleichen
43F3   C9 80      CMP #$80       Befehl erkannt ?
43F5   F0 1B      BEQ $4412      Ja, Ende
```

```
43F7   B1 24      LDA ($24),Y  Momentanen Befehl überlesen
43F9   30 03      BMI $43FE    Wenn Ende, Skip
43FB   C8         INY
43FC   D0 F9      BNE $43F7    Unbedingter Sprung
43FE   C8         INY
43FF   E6 0D      INC $0D      Befehlszähler erhöhen
4401   18         CLC
4402   98         TYA          Zeiger auf Tabelle erhöhen
4403   65 24      ADC $24
4405   85 24      STA $24
4407   90 02      BCC $440B
4409   E6 25      INC $25
440B   18         CLC          CLC = Befehl nicht gefunden
440C   A0 00      LDY #$00
440E   B1 24      LDA ($24),Y  Tabellenende ?
4410   D0 DA      BNE $43EC    Nein, weitersuchen
4412   05 0D      ORA $0D      Tokennummer setzen
4414   85 0D      STA $0D
4416   60         RTS

****************************** Befehlsklartexte

4417   45 4E C4 46 4F D2 4E 45 END FOR NEXT
441F   58 D4 44 41 54 C1 49 4E DATA INPUT#
4427   50 55 54 A3 49 4E 50 55 INPUT
442F   D4 44 49 CD 52 45 41 C4 DIM READ
4437   4C 45 D4 47 4F 54 CF 52 LET GOTO RUN
443F   55 CE 49 C6 52 45 53 54 IF RESTORE
4447   4F 52 C5 47 4F 53 55 C2 GOSUB
444F   52 45 54 55 52 CE 52 45 RETURN REM
4457   CD 53 54 4F D0 4F CE 57 STOP ON WAIT
445F   41 49 D4 4C 4F 41 C4 53 LOAD SAVE
4467   41 56 C5 56 45 52 49 46 VERIFY
446F   D9 44 45 C6 50 4F 4B C5 DEF POKE
4477   50 52 49 4E 54 A3 50 52 PRINT# PRINT
447F   49 4E D4 43 4F 4E D4 4C CONT LIST
4487   49 53 D4 43 4C D2 43 4D CLR CMD
448F   C4 53 59 D3 4F 50 45 CE SYS OPEN
4497   43 4C 4F 53 C5 47 45 D4 CLOSE GET
449F   4E 45 D7 54 41 42 A8 54 NEW TAB( TO
44A7   CF 46 CE 53 50 43 A8 54 FN SPC( THEN
44AF   48 45 CE 4E 4F D4 53 54 NOT STEP
44B7   45 D0 AB AD AA AF DE 41 + - **/ ^ AND
44BF   4E C4 4F D2 BE BD BC 53 OR > = < SGN
44C7   47 CE 49 4E D4 41 42 D3 INT ABS
44CF   55 53 D2 46 52 C5 50 4F USR FRE POS
44D7   D3 53 51 D2 52 4E C4 4C SQR RND LOG
44DF   4F C7 45 58 D0 43 4F D3 EXP COS
```

```
44E7    53 49 CE 54 41 CE 41 54 SIN TAN ATN
44EF    CE 50 45 45 CB 4C 45 CE PEEK LEN
44F7    53 54 52 A4 56 41 CC 41 STR$ VAL ASC
44FF    53 C3 43 48 52 A4 4C 45 CHR$ LEFT$
4507    46 54 A4 52 49 47 48 54 RIGHT$
450F    A4 4D 49 44 A4 47 CF 52 MID$ GO RGR
4517    47 D2 52 43 4C D2 80    RCLR

451E    4A 4F D9 52 44 4F D4 44 JOY RDOT DEC
4526    45 C3 48 45 58 A4 45 52 HEX$ ERR$
452E    52 A4 49 4E 53 54 D2 45 INSTR ELSE
4536    4C 53 C5 52 45 53 55 4D RESUME
453E    C5 54 52 41 D0 54 52 4F TRAP TRON
4546    CE 54 52 4F 46 C6 53 4F TROFF SOUND
454E    55 4E C4 56 4F CC 41 55 VOL AUTO
4556    54 CF 50 55 44 45 C6 47 PUDEF GRAPHIC
455E    52 41 50 48 49 C3 50 41 PAINT
4566    49 4E D4 43 48 41 D2 42 CHAR BOX
456E    4F D8 43 49 52 43 4C C5 CIRCLE
4576    47 53 48 41 50 C5 53 53 GSHAPE SSHAPE
457E    48 41 50 C5 44 52 41 D7 DRAW
4586    4C 4F 43 41 54 C5 43 4F LOCATE COLOR
458E    4C 4F D2 53 43 4E 43 4C SCNCLR
4596    D2 53 43 41 4C C5 48 45 SCALE HELP
459E    4C D0 44 CF 4C 4F 4F D0 DO LOOP
45A6    45 58 49 D4 44 49 52 45 EXIT DIRECTORY
45AE    43 54 4F 52 D9 44 53 41 DSAVE
45B6    56 C5 44 4C 4F 41 C4 48 DLOAD HEADER
45BE    45 41 44 45 D2 53 43 52 SCRATCH
45C6    41 54 43 C8 43 4F 4C 4C COLLECT
45CE    45 43 D4 43 4F 50 D9 52 COPY RENAME
45D6    45 4E 41 4D C5 42 41 43 BACKUP
45DE    4B 55 D0 44 45 4C 45 54 DELETE
45E6    C5 52 45 4E 55 4D 42 45 RENUMBER
45EE    D2 4B 45 D9 4D 4F 4E 49 KEY MONITOR
45F6    54 4F D2 55 53 49 4E C7 USING
45FE    55 4E 54 49 CC 57 48 49 UNTIL WHILE
4606    4C C5 00

4609    42 41 4E CB 46 49 4C 54 BANK FILTER
4611    45 D2 50 4C 41 D9 54 45 PLAY TEMPO
4619    4D 50 CF 4D 4F 56 53 50 MOVSPR
4621    D2 53 50 52 49 54 C5 53 SPRITE SPRCOLOR
4629    50 52 43 4F 4C 4F D2 52 RREG
4631    52 45 C7 45 4E 56 45 4C ENVELOPE
4639    4F 50 C5 53 4C 45 45 D0 SLEEP
4641    43 41 54 41 4C 4F C7 44 CATALOG DOPEN
4649    4F 50 45 CE 41 50 50 45 APPEND
```

```
4651    4E C4 44 43 4C 4F 53 C5 DCLOSE
4659    42 53 41 56 C5 42 4C 4F BSAVE BLOAD
4661    41 C4 52 45 43 4F 52 C4 RECORD
4669    43 4F 4E 43 41 D4 44 56 CONCAT DVERIFY
4671    45 52 49 46 D9 44 43 4C DCLEAR
4679    45 41 D2 53 50 52 53 41 SPRSAV
4681    D6 43 4F 4C 4C 49 53 49 COLLISION
4689    4F CE 42 45 47 49 CE 42 BEGIN BEND
4691    45 4E C4 57 49 4E 44 4F WINDOW
4699    D7 42 4F 4F D4 57 49 44 BOOT WIDTH
46A1    54 C8 53 50 52 44 45 C6 SPRDEF
46A9    51 55 49 D4 53 54 41 53 QUIT STASH
46B1    C8 A0 46 45 54 43 C8 A0 FETCH
46B9    53 57 41 D0 4F 46 C6 46 SWAP OFF
46C1    41 53 D4 53 4C 4F D7 00 FAST SLOW

46C9    50 4F D4 42 55 4D D0 50 POT BUMP
46D1    45 CE 52 53 50 50 4F D3 PEN RSPPOS
46D9    52 53 50 52 49 54 C5 52 RSPRITE
46E1    53 50 43 4F 4C 4F D2 58 RSPCOLOR XOR
46E9    4F D2 52 57 49 4E 44 4F RWINDOW
46F1    D7 50 4F 49 4E 54 45 D2 POINTER
46F9    00

46FA    00 00

****************************** Token und Adressen der Befehle

46FC    CC 4B        $80       $4BCD END
46FE    F8 5D        $81       $5DF9 FOR
4700    F3 57        $82       $57F4 NEXT
4702    8E 52        $83       $528F DATA
4704    47 56        $84       $5648 INPUT#
4706    61 56        $85       $5662 INPUT
4708    7A 58        $86       $587B DIM
470A    A8 56        $87       $56A9 READ
470C    C5 53        $88       $53C6 LET
470E    DA 59        $89       $59DB GOTO
4710    9A 5A        $8A       $5A9B RUN
4712    C4 52        $8B       $52C5 IF
4714    C9 5A        $8C       $5ACA RESTORE
4716    CE 59        $8D       $59CF GOSUB
4718    61 52        $8E       $5262 RETURN
471A    9C 52        $8F       $529D REM
471C    CA 4B        $90       $4BCB STOP
471E    A2 53        $91       $53A3 ON
4720    2C 6C        $92       $6C2D WAIT
4722    2B 91        $93       $912C LOAD
```

4724	11 91	$94	$9112 SAVE
4726	28 91	$95	$9129 VERIFY
4728	F9 84	$96	$84FA DEF
472A	E4 80	$97	$80E5 POKE
472C	39 55	$98	$553A PRINT#
472E	59 55	$99	$555A PRINT
4730	5F 5A	$9A	$5A60 CONT
4732	E1 50	$9B	$50E2 LIST
4734	F7 51	$9C	$51F8 CLR
4736	3F 55	$9D	$5540 CMD
4738	84 58	$9E	$5885 SYS
473A	8C 91	$9F	$918D OPEN
473C	99 91	$A0	$919A CLOSE
473E	11 56	$A1	$5612 GET
4740	D5 51	$A2	$51D6 NEW
4742	90 53	$D5	$5391 ELSE
4744	61 5F	$D6	$5F62 RESUME
4746	4C 5F	$D7	$5F4D TRAP
4748	B3 58	$D8	$58B4 TRON
474A	B6 58	$D9	$58B7 TROFF
474C	EB 71	$DA	$71EC SOUND
474E	C4 71	$DB	$71C5 VOL
4750	74 59	$DC	$5975 AUTO
4752	33 5F	$DD	$5F34 PUDEF
4754	59 6B	$DE	$6B5A GRAPHIC
4756	A7 61	$DF	$61A8 PAINT
4758	D6 67	$E0	$67D7 CHAR
475A	B6 62	$E1	$62B7 BOX
475C	8D 66	$E2	$668E CIRCLE
475E	8C 65	$E3	$658D GSHAPE
4760	2A 64	$E4	$642B SSHAPE
4762	96 67	$E5	$6797 DRAW
4764	54 69	$E6	$6955 LOCATE
4766	E1 69	$E7	$69E2 COLOR
4768	78 6A	$E8	$6A79 SCNCLR
476A	5F 69	$E9	$6960 SCALE
476C	85 59	$EA	$5986 HELP
476E	DF 5F	$EB	$5FE0 DO
4770	89 60	$EC	$608A LOOP
4772	38 60	$ED	$6039 EXIT
4774	7D A0	$EE	$A07E DIRECTORY
4776	8B A1	$EF	$A18C DSAVE
4778	A6 A1	$F0	$A1A7 DLOAD
477A	66 A2	$F1	$A267 HEADER
477C	A0 A2	$F2	$A2A1 SCRATCH
477E	2E A3	$F3	$A32F COLLECT
4780	45 A3	$F4	$A346 COPY

4782	6D A3	$F5	$A36E RENAME
4784	7B A3	$F6	$A37C BACKUP
4786	86 5E	$F7	$5E87 DELETE
4788	F7 5A	$F8	$5AF8 RENUMBER
478A	09 61	$F9	$610A KEY
478C	FF AF	$FA	$B000 MONITOR
478E	C8 6B	$FE $02	$6BC9 BANK
4790	45 70	$FE $03	$7046 FILTER
4792	E0 6D	$FE $04	$6DE1 PLAY
4794	D6 6F	$FE $05	$6FD7 TEMPO
4796	C5 6C	$FE $06	$6CC6 MOVSPR
4798	4E 6C	$FE $07	$6C4F SPRITE
479A	8F 71	$FE $08	$7190 SPRCOLOR
479C	BC 58	$FE $09	$58BD RREG
479E	C0 70	$FE $0A	$70C1 ENVELOPE
47A0	D6 6B	$FE $0B	$6BD7 SLEEP
47A2	7D A0	$FE $0C	$A07E CATALOG
47A4	1C A1	$FE $0D	$A11D DOPEN
47A6	33 A1	$FE $0E	$A134 APPEND
47A8	6E A1	$FE $0F	$A16F DCLOSE
47AA	C7 A1	$FE $10	$A1C8 BSAVE
47AC	17 A2	$FE $11	$A218 BLOAD
47AE	D6 A2	$FE $12	$A2D7 RECORD
47B0	61 A3	$FE $13	$A362 CONCAT
47B2	A3 A1	$FE $14	$A1A4 DVERIFY
47B4	21 A3	$FE $15	$A322 DCLEAR
47B6	EB 76	$FE $16	$76EC SPRSAV
47B8	63 71	$FE $17	$7164 COLLISION
47BA	6B 79	$FE $18	$796C BEGIN
47BC	8E 52	$FE $19	$528F BEND
47BE	CB 72	$FE $1A	$72CC WINDOW
47C0	34 73	$FE $1B	$7335 BOOT
47C2	B5 71	$FE $1C	$71B6 WIDTH
47C4	71 73	$FE $1D	$7372 SPRDEF
47C6	45 48	$FE $1E	$4846 QUIT
47C8	1E AA	$FE $1F	$AA1F STASH
47CA	00 00	$FE $20	$0000 ' '
47CC	23 AA	$FE $21	$AA24 FETCH
47CE	00 00	$FE $22	$0000 ' '
47D0	28 AA	$FE $23	$AA29 SWAP
47D2	45 48	$FE $24	$4846 OFF
47D4	B2 77	$FE $25	$77B3 FAST
47D6	C3 77	$FE $26	$77C4 SLOW
47D8	65 8C	$B4	$8C65 SGN
47DA	FB 8C	$B5	$8CFB INT
47DC	84 8C	$B6	$8C84 ABS

```
47DE    18 12          $B7        $1218 USR
47E0    00 80          $B8        $8000 FRE
47E2    D0 84          $B9        $84D0 POS
47E4    B7 8F          $BA        $8FB7 SQR
47E6    34 84          $BB        $8434 RND
47E8    CA 89          $BC        $89CA LOG
47EA    33 90          $BD        $9033 EXP
47EC    09 94          $BE        $9409 COS
47EE    10 94          $BF        $9410 SIN
47F0    59 94          $C0        $9459 TAN
47F2    B3 94          $C1        $94B3 ATN
47F4    C5 80          $C2        $80C5 PEEK
47F6    68 86          $C3        $8668 LEN
47F8    AE 85          $C4        $85AE STR$
47FA    4A 80          $C5        $804A VAL
47FC    77 86          $C6        $8677 ASC
47FE    BF 85          $C7        $85BF CHR$
4800    D6 85          $C8        $85D6 LEFT$
4802    0A 86          $C9        $860A RIGHT$
4804    1C 86          $CA        $861C MID$

4806    82 81          $CC        $8182 RGR
4808    9B 81          $CD        $819B RCLR
480A    00 00                     $0000
480C    03 82          $CF        $8203 JOY
480E    0C 9B          $D0        $9B0C RDOT
4810    76 80          $D1        $8076 DEC
4812    42 81          $D2        $8142 HEX$
4814    F6 80          $D3        $80F6 ERR$

4816    4D 82          $CE $02    $824D POT
4818    7C 83          $CE $03    $837C BUMP
481A    AE 82          $CE $04    $82AE PEN
481C    97 83          $CE $05    $8397 RSPPOS
481E    1E 83          $CE $06    $831E RSPRITE
4820    61 83          $CE $07    $8361 RSPCOLOR
4822    E1 83          $CE $08    $83E1 XOR
4824    07 84          $CE $09    $8407 RWINDOW
4826    FA 82          $CE $0A    $82FA POINTER

****************************** Hierarchiecodes, Token und Adressen
                               der Operatoren

4828    79 47 88  $79 $AA         $8848 +
482B    79 30 88  $79 $AB         $8831 -
482E    7B 26 8A  $7B $AC         $8A27 *
4831    7B 4B 8B  $7B $AD         $8B4C /
4834    7F C0 8F  $7F $AE         $8FC1 ^
```

```
4837   50 88 4C  $50 $AF        $4C89 AND
483A   46 85 4C  $46 $B0        $4C86 OR
483D   7D F9 8F  $7D            $8FFA Vorzeichenwechsel
4840   5A 2F 79  $5A $A8        $7930 NOT
4843   64 B5 4C  $64 $B2        $4CB6 Vergleich

****************************** Unimplemented Command

4846   A2 28      LDX #$28      'UNIMPLEMENTED COMMAND'
4848   4C 3C 4D   JMP $4D3C

****************************** Fehlermeldungstexte

484B   54 4F 4F 20 4D 41 4E 59  1  'TOO MANY FILES'
4853   20 46 49 4C 45 D3 46 49  2  'FILE OPEN'
485B   4C 45 20 4F 50 45 CE 46  3  'FILE NOT OPEN'
4863   49 4C 45 20 4E 4F 54 20
486B   4F 50 45 CE 46 49 4C 45  4  'FILE NOT FOUND'
4873   20 4E 4F 54 20 46 4F 55
487B   4E C4 44 45 56 49 43 45  5  'DEVICE NOT PRESENT'
4883   20 4E 4F 54 20 50 52 45
488B   53 45 4E D4 4E 4F 54 20  6  'NOT INPUT FILE'
4893   49 4E 50 55 54 20 46 49
489B   4C C5 4E 4F 54 20 4F 55  7  'NOT OUTPUT FILE'
48A3   54 50 55 54 20 46 49 4C
48AB   C5 4D 49 53 53 49 4E 47  8  'MISSING FILE NAME'
48B3   20 46 49 4C 45 20 4E 41
48BB   4D C5 49 4C 4C 45 47 41  9  'ILLEGAL DEVICE NUMBER'
48C3   4C 20 44 45 56 49 43 45
48CB   20 4E 55 4D 42 45 D2 4E  10 'NEXT WITHOUT FOR'
48d3   45 58 54 20 57 49 54 48
48DB   4F 55 54 20 46 4F D2 53  11 'SYNTAX'
48E3   59 4E 54 41 D8 52 45 54  12 'RETURN WITHOUT GOSUB'
48EB   55 52 4E 20 57 49 54 48
48F3   4F 55 54 20 47 4F 53 55
48FB   C2 4F 55 54 20 4F 46 20  13 'OUT OF DATA'
4903   44 41 54 C1 49 4C 4C 45  14 'ILLEGAL QUANTITY'
490B   47 41 4C 20 51 55 41 4E
4913   54 49 54 D9 4F 56 45 52  15 'OVERFLOW'
491B   46 4C 4F D7 4F 55 54 20  16 'OUT OF MEMORY'
4923   4F 46 20 4D 45 4D 4F 52
492B   D9 55 4E 44 45 46 27 44  17 'UNDEF'D STATEMENT'
4933   20 53 54 41 54 45 4D 45
493B   4E D4 42 41 44 20 53 55  18 'BAD SUBSCRIPT'
4943   42 53 43 52 49 50 D4 52  19 'REDIM'D ARRAY'
494B   45 44 49 4D 27 44 20 41
4953   52 52 41 D9 44 49 56 49  20 'DIVISION BY ZERO'
495B   53 49 4F 4E 20 42 59 20
```

```
4963    5A 45 52 CF 49 4C 4C 45   21 'ILLEGAL DIRECT'
496B    47 41 4C 20 44 49 52 45
4973    43 D4 54 59 50 45 20 4D   22 'TYPE MISMATCH'
497B    49 53 4D 41 54 43 C8 53   23 'STRING TOO LONG'
4983    54 52 49 4E 47 20 54 4F
498B    4F 20 4C 4F 4E C7 46 49   24 'FILE DATA'
4993    4C 45 20 44 41 54 C1 46   25 'FORMULA TOO COMPLEX'
499B    4F 52 4D 55 4C 41 20 54
49A3    4F 4F 20 43 4F 4D 50 4C
49AB    45 D8 43 41 4E 27 54 20   26 'CAN'T CONTINUE'
49B3    43 4F 4E 54 49 4E 55 C5
49BB    55 4E 44 45 46 27 44 20   27 'UNDEF'D FUNCTION'
49C3    46 55 4E 43 54 49 4F CE
49CB    56 45 52 49 46 D9 4C 4F   28 'VERIFY'
49D3    41 C4 42 52 45 41 4B 00   29 'LOAD'     30 'BREAK'
49DB    A0 43 41 4E 27 54 20 52   31 'CAN'T RESUME'
49E3    45 53 55 4D C5 4C 4F 4F   32 'LOOP NOT FOUND'
49EB    50 20 4E 4F 54 20 46 4F
49F3    55 4E C4 4C 4F 4F 50 20   33 'LOOP WITHOUT DO'
49FB    57 49 54 48 4F 55 54 20
4A03    44 CF 44 49 52 45 43 54   34 'DIRECT MODE ONLY'
4A0B    20 4D 4F 44 45 20 4F 4E
4A13    4C D9 4E 4F 20 47 52 41   35 'NO GRAPHICS AREA'
4A1B    50 48 49 43 53 20 41 52
4A23    45 C1 42 41 44 20 44 49   36 'BAD DISK'
4A2B    53 CB 42 45 4E 44 20 4E   37 'BEND NOT FOUND'
4A33    4F 54 20 46 4F 55 4E C4
4A3B    4C 49 4E 45 20 4E 55 4D   38 'LINE NUMBER TOO LARGE'
4A43    42 45 52 20 54 4F 4F 20
4A4B    4C 41 52 47 C5 55 4E 52   39 'UNRESOLVED REFERENCE'
4A53    45 53 4F 4C 56 45 44 20
4A5B    52 45 46 45 52 45 4E 43
4A63    C5 55 4E 49 4D 50 4C 45   40 'UNIMPLEMENTED COMMAND'
4A6B    4D 45 4E 54 45 44 20 43
4A73    4F 4D 4D 41 4E C4 46 49   41 'FILE READ'
4A7B    4C 45 20 52 45 41 C4

****************************** Zeiger auf Fehlermeldung
                               Nummer (X) setzen

4A82    AA          TAX           Nummer in (X)
4A83    A0 00       LDY #$00
4A85    A9 4B       LDA #$4B      Fehlermeldungstexte
4A87    85 26       STA $26       ab $484B
4A89    A9 48       LDA #$48
4A8B    85 27       STA $27
4A8D    CA          DEX           Fehlertext erreicht ?
4A8E    30 0E       BMI $4A9E     Ja, Ende
```

```
4A90   B1 26      LDA ($26),Y  Zeichen holen
4A92   48         PHA
4A93   E6 26      INC $26      Pointer erhöhen
4A95   D0 02      BNE $4A99
4A97   E6 27      INC $27
4A99   68         PLA          Zeichen war Ende einer Meldung ?
4A9A   10 F4      BPL $4A90    Nein, weiterüberlesen
4A9C   30 EF      BMI $4A8D    Nächste Meldung lesen
4A9E   60         RTS

****************************** Interpreterschleife / Befehl auswerten

4A9F   6C 08 03   JMP ($0308)  $4AA2 Befehl ausführen
4AA2   24 7F      BIT $7F      Direktmodus ?
4AA4   10 4A      BPL $4AF0    Ja, Skip
4AA6   AD 7F 12   LDA $127F    BASIC-IRQ erlaubt ?
4AA9   30 45      BMI $4AF0    Nein, Skip
4AAB   A2 02      LDX #$02
4AAD   BD 76 12   LDA $1276,X  Auslöse-IRQ gefunden ?
4AB0   F0 3B      BEQ $4AED    Nein, Skip
4AB2   A9 00      LDA #$00     IRQ löschen
4AB4   9D 76 12   STA $1276,X
4AB7   BD 79 12   LDA $1279,X  Zeilennummer setzen
4ABA   85 16      STA $16
4ABC   BD 7C 12   LDA $127C,X
4ABF   85 17      STA $17
4AC1   8A         TXA
4AC2   48         PHA
4AC3   A5 3D      LDA $3D      PC retten
4AC5   48         PHA
4AC6   A5 3E      LDA $3E
4AC8   48         PHA
4AC9   AD 7F 12   LDA $127F    BASIC-IRQ verbieten
4ACC   09 80      ORA #$80
4ACE   8D 7F 12   STA $127F
4AD1   20 80 03   JSR $0380    CHRGET
4AD4   20 1D 5A   JSR $5A1D    GOSUB-Werte auf den BASIC-Stack
4AD7   20 E2 59   JSR $59E2    GOTO-Befehl ausführen
4ADA   20 F6 4A   JSR $4AF6    BASIC-IRQ-Programm ausführen
4ADD   AD 7F 12   LDA $127F    BASIC-IRQ wieder erlauben
4AE0   29 7F      AND #$7F
4AE2   8D 7F 12   STA $127F
4AE5   68         PLA          PC holen
4AE6   85 3E      STA $3E
4AE8   68         PLA
4AE9   85 3D      STA $3D
4AEB   68         PLA
4AEC   AA         TAX
```

```
4AED   CA         DEX         Alle IRQ-Quellen abgearbeitet ?
4AEE   10 BD      BPL $4AAD   Nein, weitertesten
4AF0   20 80 03   JSR $0380   CHRGET
4AF3   20 3F 4B   JSR $4B3F   Befehl ausführen

****************************** Interpreterschleifenstart

4AF6   20 B5 4B   JSR $4BB5   Test auf STOP-Taste
4AF9   24 7F      BIT $7F     Direktmodus ?
4AFB   10 06      BPL $4B03   Ja, Skip
4AFD   20 34 4B   JSR $4B34   PC für CONT setzen
4B00   BA         TSX         Stapelzeiger retten
4B01   86 82      STX $82
4B03   A0 00      LDY #$00
4B05   20 C9 03   JSR $03C9   Aktuelles Zeichen holen
4B08   F0 03      BEQ $4B0D   Wenn Zeilenende, Skip
4B0A   4C AE 4B   JMP $4BAE   Test auf ':'
4B0D   24 7F      BIT $7F     Direktmodus ?
4B0F   10 20      BPL $4B31   Ja, Skip
4B11   A0 02      LDY #$02
4B13   20 C9 03   JSR $03C9   Programmende ?
4B16   F0 19      BEQ $4B31   Ja, 'READY.'
4B18   C8         INY
4B19   20 C9 03   JSR $03C9   Aktuelle Zeilennummer
4B1C   85 3B      STA $3B     setzen
4B1E   C8         INY
4B1F   20 C9 03   JSR $03C9
4B22   85 3C      STA $3C
4B24   98         TYA         PC korrigieren
4B25   18         CLC
4B26   65 3D      ADC $3D
4B28   85 3D      STA $3D
4B2A   90 02      BCC $4B2E
4B2C   E6 3E      INC $3E
4B2E   4C 9F 4A   JMP $4A9F   Interpreterschleife
4B31   4C 37 4D   JMP $4D37   'READY.'

****************************** PC für CONT setzen

4B34   A5 3D      LDA $3D     PC holen
4B36   A4 3E      LDY $3E
4B38   8D 02 12   STA $1202   und in CONT-Zeiger setzen
4B3B   8C 03 12   STY $1203
4B3E   60         RTS

****************************** Befehl ausführen

4B3F   F0 FD      BEQ $4B3E   Wenn Trennzeichen, Skip
```

```
4B41   2C 6F 11   BIT $116F     Trace ?
4B44   10 13      BPL $4B59     Nein, Skip
4B46   24 7F      BIT $7F       Direktmodus ?
4B48   10 0F      BPL $4B59     Ja, Skip
4B4A   48         PHA
4B4B   A9 5B      LDA #$5B      Eckige Klammer auf
4B4D   20 0C 56   JSR $560C     Zeichen ausgeben
4B50   20 2E 8E   JSR $8E2E     Zeilennummer ausgeben
4B53   A9 5D      LDA #$5D      Eckige Klammer zu
4B55   20 0C 56   JSR $560C     Zeichen ausgeben
4B58   68         PLA
4B59   C9 FE      CMP #$FE      Sondertokenwert 1 ?
4B5B   F0 37      BEQ $4B94     Ja, Skip
4B5D   C9 CB      CMP #$CB      Token für GO ?
4B5F   D0 03      BNE $4B64     Nein, Skip
4B61   4C 3D 5A   JMP $5A3D     GO64 und GO TO Test
4B64   C9 CA      CMP #$CA      Token für MID$ ?
4B66   F0 23      BEQ $4B8B     Ja, ausführen
4B68   C9 FB      CMP #$FB      Ungültiger Tokenwert ?
4B6A   B0 3F      BCS $4BAB     Ja, Fehler
4B6C   C9 A3      CMP #$A3
4B6E   90 06      BCC $4B76
4B70   C9 D5      CMP #$D5
4B72   90 37      BCC $4BAB     Ja, Fehler
4B74   E9 32      SBC #$32      Tokenwert anpassen
4B76   38         SEC
4B77   E9 80      SBC #$80      Tokengrundwert $80 abziehen
4B79   B0 03      BCS $4B7E     Wenn gültiges Token, Skip
4B7B   4C C6 53   JMP $53C6     LET-Befehl
4B7E   0A         ASL           Token in Offset umrechnen
4B7F   A8         TAY
4B80   B9 FD 46   LDA $46FD,Y   Befehlsadresse auf Stapel legen
4B83   48         PHA
4B84   B9 FC 46   LDA $46FC,Y
4B87   48         PHA
4B88   4C 80 03   JMP $0380     CHRGET und Befehlsaufruf

4B8B   A9 59      LDA #$59      MID$ ausführen ($5901)
4B8D   48         PHA
4B8E   A9 00      LDA #$00
4B90   48         PHA
4B91   4C 80 03   JMP $0380     CHRGET

4B94   20 80 03   JSR $0380     Sondertoken testen und
4B97   F0 12      BEQ $4BAB     ausführen
4B99   C9 02      CMP #$02      Tokenwert ungültig ?
4B9B   90 08      BCC $4BA5     Ja, Escapetest
4B9D   C9 27      CMP #$27
```

```
4B9F   B0 04      BCS $4BA5     Ja, Escapetest
4BA1   69 47      ADC #$47      Tokenwert anpassen
4BA3   D0 D9      BNE $4B7E     Unbedingter Sprung

4BA5   38         SEC           Fehlerflag vorbesetzen
4BA6   6C 10 03   JMP ($0310)   $4BA9 ESC-Befehl ausführen
4BA9   90 E6      BCC $4B91     Wenn kein Fehler, Befehlsaufruf
4BAB   4C 6C 79   JMP $796C     'SYNTAX'

4BAE   C9 3A      CMP #$3A      ':' ?
4BB0   D0 F9      BNE $4BAB     Nein, Fehler
4BB2   4C 9F 4A   JMP $4A9F     Zurück zur Befehlsausführung

****************************** STOP-Taste testen

4BB5   20 93 92   JSR $9293     STOP-Taste gedrückt ?
4BB8   F0 01      BEQ $4BBB     Ja, Skip
4BBA   60         RTS
4BBB   AC 0C 12   LDY $120C     TRAP erlaubt ?
4BBE   C8         INY
4BBF   F0 0F      BEQ $4BD0     Nein, Skip
4BC1   20 93 92   JSR $9293     Warten bis Taste losgelassen
4BC4   F0 FB      BEQ $4BC1
4BC6   A2 1E      LDX #$1E      'BREAK'
4BC8   4C 3C 4D   JMP $4D3C

****************************** BASIC-Befehl STOP

4BCB   B0 01      BCS $4BCE     Unbedingter Sprung

****************************** BASIC-Befehl END

4BCD   18         CLC           END-Flag setzen
4BCE   D0 26      BNE $4BF6     Wenn kein Trennzeichen, Ende
4BD0   24 7F      BIT $7F       Direktmodus ?
4BD2   10 0D      BPL $4BE1     Ja,Skip
4BD4   20 34 4B   JSR $4B34     PC für CONT setzen
4BD7   A5 3B      LDA $3B       Zeilennummer retten
4BD9   A4 3C      LDY $3C
4BDB   8D 00 12   STA $1200
4BDE   8C 01 12   STY $1201
4BE1   68         PLA           Rücksprungadresse löschen
4BE2   68         PLA
4BE3   90 0E      BCC $4BF3     Bei Aufruf durch END, Skip
4BE5   20 81 92   JSR $9281     Text ausgeben

****************************** Textkonstante 'BREAK'
```

```
4BE8   0D 0A 42 52 45 41 4B 00 (CR) (LF) 'BREAK'

******************************

4BF0   4C AF 4D   JMP $4DAF    'IN' Nummer, 'READY.'
4BF3   4C 37 4D   JMP $4D37    'READY.'
4BF6   60         RTS

****************************** Funktion aufrufen

4BF7   C9 CE      CMP #$CE     Sondertoken 2 ?
4BF9   F0 59      BEQ $4C54    Ja, Skip
4BFB   C9 D5      CMP #$D5     Keine Funktion ?
4BFD   B0 AC      BCS $4BAB    Ja, Fehler
4BFF   C9 CB      CMP #$CB     BASIC 2.0 Funktion ?
4C01   90 02      BCC $4C05    Ja, Skip
4C03   E9 01      SBC #$01     BASIC 7.0 Funktionstoken anpassen
4C05   48         PHA          Wert retten
4C06   AA         TAX          und in (X)
4C07   20 80 03   JSR $0380    CHRGET
4C0A   E0 D3      CPX #$D3     Funktion ERR$ ?
4C0C   F0 08      BEQ $4C16    Ja, Skip
4C0E   E0 CB      CPX #$CB     BASIC 7.0 Funktion ?
4C10   B0 29      BCS $4C3B    Ja, Skip
4C12   E0 C8      CPX #$C8     Numerische Funktion ?
4C14   90 25      BCC $4C3B    Ja, Skip
4C16   20 59 79   JSR $7959    Test auf '('
4C19   20 EF 77   JSR $77EF    FRMEVL
4C1C   20 5C 79   JSR $795C    Test auf ')'
4C1F   20 DD 77   JSR $77DD    Test auf String
4C22   68         PLA
4C23   C9 D3      CMP #$D3     Funktion INSTR ?
4C25   F0 59      BEQ $4C80    Ja, Sonderbehandlung
4C27   AA         TAX          Token retten
4C28   A5 67      LDA $67      Deskriptoradresse retten
4C2A   48         PHA
4C2B   A5 66      LDA $66
4C2D   48         PHA
4C2E   8A         TXA          Tokenwert auf Stapel
4C2F   48         PHA
4C30   20 F4 87   JSR $87F4    Byte-Wert für Funktion auswerten
4C33   68         PLA          Tokenwert wieder holen
4C34   A8         TAY          und in (Y) retten
4C35   8A         TXA          Byte-Wert auf Stapel legen
4C36   48         PHA
4C37   98         TYA          Tokenwert wieder in (A)
4C38   4C 3F 4C   JMP $4C3F
```

```
4C3B   20 50 79   JSR $7950     Term in Klammern auswerten
4C3E   68         PLA           Token wieder holen
4C3F   38         SEC
4C40   E9 B4      SBC #$B4      in Offset umrechnen
4C42   0A         ASL
4C43   A8         TAY
4C44   B9 D9 47   LDA $47D9,Y   Funktionsadresse setzen
4C47   85 58      STA $58
4C49   B9 D8 47   LDA $47D8,Y
4C4C   85 57      STA $57
4C4E   20 56 00   JSR $0056     Funktion aufrufen
4C51   4C DA 77   JMP $77DA     und Ergebnis auf numerisch prüfen

4C54   20 80 03   JSR $0380     CHRGET
4C57   F0 2A      BEQ $4C83     Wenn Trennzeichen, Fehler
4C59   C9 0A      CMP #$0A      Funktion POINTER ?
4C5B   F0 0B      BEQ $4C68     Ja, Skip
4C5D   48         PHA           Token retten
4C5E   20 80 03   JSR $0380     CHRGET
4C61   20 59 79   JSR $7959     Test auf '('
4C64   20 EF 77   JSR $77EF     FRMEVL
4C67   68         PLA           Token wieder holen
4C68   C9 02      CMP #$02      Ungültiges Token ?
4C6A   90 08      BCC $4C74     Ja, Sonderbehandlung
4C6C   C9 0B      CMP #$0B
4C6E   B0 04      BCS $4C74     Ja, Sonderbehandlung
4C70   69 D1      ADC #$D1      Tabellenoffset berechnen
4C72   D0 CB      BNE $4C3F     Unbedingter Sprung

4C74   38         SEC           Fehlerflag setzen
4C75   20 7D 4C   JSR $4C7D     Sonderfunktion aufrufen
4C78   B0 09      BCS $4C83     Wenn Fehler, Skip
4C7A   4C DA 77   JMP $77DA     Ergebnis auf numerisch prüfen
4C7D   6C FC 02   JMP ($02FC)   ESC-Funktionsaufruf
4C80   4C C1 99   JMP $99C1     INSTR aufrufen
4C83   4C 6C 79   JMP $796C     'SYNTAX'

******************************** BASIC-Befehl OR

4C86   A0 FF      LDY #$FF      Verknüpfungswert laden
4C88   2C         .BYTE $2C

******************************** BASIC-Befehl AND

4C89   A0 00      LDY #$00      Verknüpfungswert laden
4C8B   84 0D      STY $0D
4C8D   20 B4 84   JSR $84B4     FAC#1 in Integer
4C90   A5 66      LDA $66       Werte verknüpfen
```

```
4C92   45 0D      EOR $0D
4C94   85 09      STA $09
4C96   A5 67      LDA $67
4C98   45 0D      EOR $0D
4C9A   85 0A      STA $0A
4C9C   20 28 8C   JSR $8C28    FAC#2 in FAC#1
4C9F   20 B4 84   JSR $84B4    FAC#1 in Integer
4CA2   A5 67      LDA $67      Ergebnis setzen
4CA4   45 0D      EOR $0D
4CA6   25 0A      AND $0A
4CA8   45 0D      EOR $0D
4CAA   A8         TAY
4CAB   A5 66      LDA $66
4CAD   45 0D      EOR $0D
4CAF   25 09      AND $09
4CB1   45 0D      EOR $0D
4CB3   4C 3C 79   JMP $793C    Wert in FAC#1

******************************** Vergleich

4CB6   20 DE 77   JSR $77DE    Auf gleichen Typ testen
4CB9   B0 13      BCS $4CCE    Wenn String, Skip
4CBB   A5 6F      LDA $6F
4CBD   09 7F      ORA #$7F     Speicherformat setzen
4CBF   25 6B      AND $6B
4CC1   85 6B      STA $6B
4CC3   A9 6A      LDA #$6A     Zeiger auf FAC#2
4CC5   A0 00      LDY #$00
4CC7   20 87 8C   JSR $8C87    Vergleich Konstante mit FAC#1
4CCA   AA         TAX
4CCB   4C 01 4D   JMP $4D01

******************************** Stringvergleich

4CCE   A9 00      LDA #$00     Typ auf numerisch setzen
4CD0   85 0F      STA $0F
4CD2   C6 4F      DEC $4F      Vergleichsmaske anpassen
4CD4   20 81 87   JSR $8781    Stringparameter holen + FRESTR
4CD7   85 63      STA $63      Deskriptor speichern
4CD9   86 64      STX $64
4CDB   84 65      STY $65
4CDD   A5 6D      LDA $6D      Zeiger auf 2. String
4CDF   A4 6E      LDY $6E
4CE1   20 85 87   JSR $8785    FRESTR
4CE4   86 6D      STX $6D      Adresse setzen
4CE6   84 6E      STY $6E
4CE8   AA         TAX          Länge des 2. Strings in (X)
4CE9   38         SEC
```

```
4CEA   E5 63      SBC $63      Stringlängen gleich ?
4CEC   F0 08      BEQ $4CF6    Ja, Skip
4CEE   A9 01      LDA #$01     Flag für 2. String kürzer setzen
4CF0   90 04      BCC $4CF6    Wenn 2. String kürzer, Skip
4CF2   A6 63      LDX $63      Länge des 1. Strings in (X)
4CF4   A9 FF      LDA #$FF     Flag für 1. String kürzer
4CF6   85 68      STA $68      setzen
4CF8   A0 FF      LDY #$FF     Offset auf Start
4CFA   E8         INX
4CFB   C8         INY
4CFC   CA         DEX          Ende erreicht ?
4CFD   D0 07      BNE $4D06    Nein, Skip
4CFF   A6 68      LDX $68      1. String kürzer ?

4D01   30 1B      BMI $4D1E    Ja, Skip
4D03   18         CLC          Flag setzen
4D04   90 18      BCC $4D1E    Unbedingter Sprung

4D06   A9 6D      LDA #$6D     Zeichen aus 2. String holen
4D08   20 AB 03   JSR $03AB
4D0B   48         PHA          und retten
4D0C   A9 64      LDA #$64
4D0E   20 AB 03   JSR $03AB    Zeichen aus 1. String holen
4D11   85 79      STA $79
4D13   68         PLA
4D14   C5 79      CMP $79      Zeichen gleich ?
4D16   F0 E3      BEQ $4CFB    Ja, weitervergleichen
4D18   A2 FF      LDX #$FF     Flag für 1. String kürzer laden
4D1A   B0 02      BCS $4D1E    Wenn kürzer, Skip
4D1C   A2 01      LDX #$01     Flag für 2. String kürzer laden
4D1E   E8         INX          Vergleichsergebnis setzen
4D1F   8A         TXA
4D20   2A         ROL
4D21   25 14      AND $14
4D23   F0 02      BEQ $4D27
4D25   A9 FF      LDA #$FF
4D27   4C 68 8C   JMP $8C68    Ergebnis setzen

******************************* 'READY.' ausgeben

4D2A   20 81 92   JSR $9281    Text ausgeben

4D2D   0D 52 45 41 44 59 2E 0D (CR) 'READY.'
4D35   00

4D36   60         RTS
```

```
****************************** 'READY.'

4D37  A2 80     LDX #$80      Kein Fehler, 'READY.'-Wert
4D39  2C        .BYTE $2C

4D3A  A2 10     LDX #$10      'OUT OF MEMORY'

4D3C  6C 00 03  JMP ($0300)   $4D3F Fehlerroutine, Nummer in (X)
4D3F  8D 03 FF  STA $FF03     Write in Bank 0 setzen
4D42  8A        TXA           Fehler gesetzt ?
4D43  30 72     BMI $4DB7     Nein, Skip
4D45  8E 08 12  STX $1208     Fehlernummer für TRAP setzen
4D48  24 7F     BIT $7F       Direktmodus ?
4D4A  10 30     BPL $4D7C     Ja, Skip
4D4C  A0 01     LDY #$01      Werte für TRAP setzen
4D4E  B9 3B 00  LDA $003B,Y   BASIC-Zeilennummer setzen
4D51  99 09 12  STA $1209,Y
4D54  B9 02 12  LDA $1202,Y   PC setzen
4D57  99 0E 12  STA $120E,Y
4D5A  88        DEY
4D5B  10 F1     BPL $4D4E
4D5D  AC 0C 12  LDY $120C     TRAP erlaubt ?
4D60  C8        INY
4D61  F0 19     BEQ $4D7C     Nein, Skip
4D63  88        DEY
4D64  84 17     STY $17       Zeilennummer High setzen
4D66  8C 0D 12  STY $120D
4D69  AC 0B 12  LDY $120B     Zeilennummer Low setzen
4D6C  84 16     STY $16
4D6E  A2 FF     LDX #$FF      TRAP verbieten
4D70  8E 0C 12  STX $120C
4D73  A6 82     LDX $82       Stapelzeiger setzen
4D75  9A        TXS
4D76  20 FB 59  JSR $59FB     GOTO-Befehl aufrufen
4D79  4C F6 4A  JMP $4AF6     Interpreterschleife
4D7C  CA        DEX
4D7D  8A        TXA
4D7E  20 82 4A  JSR $4A82     Fehlertext anwählen
4D81  20 6F 92  JSR $926F     CLRCH I/O rücksetzen
4D84  A9 00     LDA #$00      Aktives I/O-Gerät = Tastatur
4D86  85 15     STA $15
4D88  24 D7     BIT $D7       80-Zeichen-Modus ?
4D8A  30 02     BMI $4D8E     Ja, Skip
4D8C  85 D8     STA $D8       Textschirm einschalten
4D8E  20 98 55  JSR $5598     (CR) out
4D91  20 0A 56  JSR $560A     '?' out
4D94  A0 00     LDY #$00      Fehlermeldungstext ausgeben
4D96  B1 26     LDA ($26),Y
```

```
4D98   48          PHA
4D99   29 7F       AND #$7F
4D9B   20 0C 56    JSR $560C     Zeichen ausgeben
4D9E   C8          INY
4D9F   68          PLA
4DA0    10 F4        BPL $4D96      Wenn Textende nicht erreicht, weiter
ausgeben
4DA2   20 38 52    JSR $5238     BASIC-Zeiger rücksetzen
4DA5   20 81 92    JSR $9281     Text ausgeben

4DA8   20 45 52 52 4F 52 0D 00 ' ERROR' (CR)

4DAF   A4 3C       LDY $3C       Zeilennummer gesetzt ?
4DB1   C8          INY
4DB2   F0 03       BEQ $4DB7     Nein, Skip
4DB4   20 26 8E    JSR $8E26     'IN' Zeilennummer ausgeben
4DB7   20 2A 4D    JSR $4D2A     'READY.' ausgeben
4DBA   A9 80       LDA #$80      Direktmodus setzen
4DBC   20 90 FF    JSR $FF90
4DBF   A9 00       LDA #$00      BASIC-Direktmodus setzen
4DC1   85 7F       STA $7F
4DC3   6C 02 03    JMP ($0302)   $4DC6 BASIC-Warmstart
4DC6   A2 FF       LDX #$FF      Zeilennummer löschen
4DC8   86 3C       STX $3C
4DCA   20 93 4F    JSR $4F93     Befehlszeile in Puffer
4DCD   86 3D       STX $3D       Zeiger auf Puffer
4DCF   84 3E       STY $3E
4DD1   20 80 03    JSR $0380     CHRGET
4DD4   AA          TAX
4DD5   F0 EC       BEQ $4DC3     Wenn Puffer leer, dann nochmal
4DD7   90 09       BCC $4DE2     Wenn erstes Zeichen Zahl, Zeile einfügen
4DD9   20 0A 43    JSR $430A     Umwandlung in Interpretercode
4DDC   20 86 03    JSR $0386     CHRGOT
4DDF   4C F3 4A    JMP $4AF3     Interpreterschleife

****************************** Löschen und Einfügen von Programmzeilen

4DE2   20 A0 50    JSR $50A0     Zeilennummer holen
4DE5   20 0A 43    JSR $430A     Umwandlung in Interpretercode
4DE8   84 0D       STY $0D       Zeilenlänge setzen
4DEA   20 64 50    JSR $5064     Zeile vorhanden ?
4DED   90 7B       BCC $4E6A     Nein, Skip
4DEF   A0 00       LDY #$00      Zeile löschen
4DF1   20 EC 42    JSR $42EC     Linkadresse Low laden
4DF4   38          SEC
4DF5   E5 61       SBC $61       - Zeilenstartadresse Low
4DF7   38          SEC
4DF8   E9 04       SBC #$04      - 4 Verwaltungsbytes
```

```
4DFA  E5 0D     SBC $0D     Alte Zeilenlänge >= benötigte Länge ?
4DFC  B0 1C     BCS $4E1A   Ja, Skip
4DFE  49 FF     EOR #$FF    Zweierkomplement
4E00  69 01     ADC #$01    ergibt notwendigen Platz
4E02  AC 11 12  LDY $1211   Nötigen Platz auf
4E05  6D 10 12  ADC $1210   Programmende aufaddieren
4E08  90 01     BCC $4E0B   Kein Übertrag, Skip
4E0A  C8        INY         High-Byte erhöhen
4E0B  CC 13 12  CPY $1213   Speicherende erreicht ?
4E0E  90 0A     BCC $4E1A   Nein, Skip
4E10  D0 05     BNE $4E17   Ja, Fehler
4E12  CD 12 12  CMP $1212
4E15  90 03     BCC $4E1A   Nein, Skip
4E17  4C 3A 4D  JMP $4D3A   'OUT OF MEMORY ERROR'
4E1A  A0 01     LDY #$01    Offset auf High-Byte der Linkadresse
4E1C  20 EC 42  JSR $42EC   High-Byte laden
4E1F  85 25     STA $25     und setzen
4E21  AD 10 12  LDA $1210   Low-Byte Programmende laden
4E24  85 24     STA $24     und als Zeigerbyte setzen
4E26  A5 62     LDA $62
4E28  85 27     STA $27
4E2A  88        DEY         Offset auf Low-Byte setzen
4E2B  20 EC 42  JSR $42EC   Low-Byte der Linkadresse laden
4E2E  18        CLC
4E2F  E5 61     SBC $61     - Startadresse der Zeile
4E31  49 FF     EOR #$FF    ergibt negative Zeilenlänge
4E33  18        CLC
4E34  6D 10 12  ADC $1210   Programmende erniedrigen
4E37  8D 10 12  STA $1210
4E3A  85 26     STA $26     und Low-Byte des Zeigers setzen
4E3C  AD 11 12  LDA $1211   High-Byte setzen
4E3F  69 FF     ADC #$FF
4E41  8D 11 12  STA $1211
4E44  E5 62     SBC $62     - High-Byte Zeilenadresse
4E46  AA        TAX         ergibt Speicherseitenzahl
4E47  38        SEC
4E48  A5 61     LDA $61     Low-Byte Zeilenadresse
4E4A  ED 10 12  SBC $1210   - Low-Byte Programmende
4E4D  A8        TAY         ergibt negativen Restbetrag
4E4E  B0 03     BCS $4E53   Kein Übertrag, Skip
4E50  E8        INX         Eine Speicherseite mehr !
4E51  C6 27     DEC $27     Zeiger korrigieren
4E53  18        CLC
4E54  65 24     ADC $24     Startzeiger
4E56  90 03     BCC $4E5B   um Restbetrag korrigieren
4E58  C6 25     DEC $25
4E5A  18        CLC
4E5B  20 05 43  JSR $4305   Zeichen des Programms holen
```

```
4E5E   91 26      STA ($26),Y  und kopieren
4E60   C8         INY          Ende der Speicherseite ?
4E61   D0 F8      BNE $4E5B    Nein, weitermachen
4E63   E6 25      INC $25      High-Bytes der Zeiger erhöhen
4E65   E6 27      INC $27
4E67   CA         DEX          Alle Speicherseiten kopiert ?
4E68   D0 F1      BNE $4E5B    Nein, weitermachen

****************************** Programmzeile einfügen

4E6A   20 38 52   JSR $5238    CLR-Befehl
4E6D   20 4F 4F   JSR $4F4F    Verkettung korrigieren
4E70   A0 00      LDY #$00
4E72   B1 3D      LDA ($3D),Y  Zeichen im Puffer ?
4E74   D0 03      BNE $4E79    Ja, Skip
4E76   4C D5 4D   JMP $4DD5    Nein, zur Eingabeschleife
4E79   18         CLC
4E7A   AD 10 12   LDA $1210    Altes Programmende
4E7D   AC 11 12   LDY $1211
4E80   85 5C      STA $5C      setzen
4E82   84 5D      STY $5D
4E84   65 0D      ADC $0D      + Länge der Eingabezeile
4E86   90 01      BCC $4E89
4E88   C8         INY
4E89   18         CLC
4E8A   69 04      ADC #$04     + 4 Verwaltungsbytes
4E8C   90 01      BCC $4E8F
4E8E   C8         INY
4E8F   85 5A      STA $5A      ergibt neues Programmende
4E91   84 5B      STY $5B
4E93   CC 13 12   CPY $1213    > Speicherende in Bank 0 ?
4E96   90 0A      BCC $4EA2    Nein, Skip
4E98   D0 05      BNE $4E9F    Ja, Fehler
4E9A   CD 12 12   CMP $1212
4E9D   90 03      BCC $4EA2    Nein, Skip
4E9F   4C 3A 4D   JMP $4D3A    'OUT OF MEMORY ERROR'
4EA2   8D 10 12   STA $1210    Neues Programmende setzen
4EA5   8C 11 12   STY $1211
4EA8   38         SEC
4EA9   A5 5C      LDA $5C      Altes Programmende Low-Byte
4EAB   E5 61      SBC $61      - Zeilenstart Low-Byte
4EAD   85 24      STA $24      ergibt Zeiger Low-Byte
4EAF   A8         TAY          und Restbetrag
4EB0   A5 5D      LDA $5D      Altes Programmende High-Byte
4EB2   E5 62      SBC $62      - Zeilenstart High-Byte
4EB4   AA         TAX
4EB5   E8         INX          ergibt Speicherseitenzahl
4EB6   98         TYA          Restbetrag ?
```

```
4EB7   F0 25      BEQ $4EDE     Nein, Skip
4EB9   A5 5C      LDA $5C       Quellzeiger korrigieren
4EBB   38         SEC
4EBC   E5 24      SBC $24
4EBE   85 5C      STA $5C
4EC0   B0 03      BCS $4EC5
4EC2   C6 5D      DEC $5D
4EC4   38         SEC           Zielzeiger korrigieren
4EC5   A5 5A      LDA $5A
4EC7   E5 24      SBC $24
4EC9   85 5A      STA $5A
4ECB   B0 09      BCS $4ED6
4ECD   C6 5B      DEC $5B
4ECF   90 05      BCC $4ED6
4ED1   20 DD 42   JSR $42DD     Zeichen aus Programm holen
4ED4   91 5A      STA ($5A),Y   und kopieren
4ED6   88         DEY           Speicherseite kopiert ?
4ED7   D0 F8      BNE $4ED1     Nein, weitermachen
4ED9   20 DD 42   JSR $42DD     Letztes Byte
4EDC   91 5A      STA ($5A),Y   kopieren
4EDE   C6 5D      DEC $5D       Seitenzeiger erniedrigen
4EE0   C6 5B      DEC $5B
4EE2   CA         DEX           Alle Speicherseiten kopiert ?
4EE3   D0 F1      BNE $4ED6     Nein, weitermachen
4EE5   A0 00      LDY #$00
4EE7   A9 01      LDA #$01
4EE9   91 61      STA ($61),Y   Linkadresse mit $0101 vorbesetzen
4EEB   C8         INY           damit Verkettungsroutine ab $4F4F
4EEC   91 61      STA ($61),Y   richtig funktioniert
4EEE   C8         INY
4EEF   A5 16      LDA $16       Zeilennummer eintragen
4EF1   91 61      STA ($61),Y
4EF3   A5 17      LDA $17
4EF5   C8         INY
4EF6   91 61      STA ($61),Y
4EF8   18         CLC
4EF9   A5 61      LDA $61       Adresse auf Programmtextanfang setzen
4EFB   69 04      ADC #$04
4EFD   85 61      STA $61
4EFF   90 02      BCC $4F03
4F01   E6 62      INC $62
4F03   A4 0D      LDY $0D       und Programmtext in Zeile eintragen
4F05   88         DEY
4F06   B1 3D      LDA ($3D),Y
4F08   91 61      STA ($61),Y
4F0A   88         DEY
4F0B   C0 FF      CPY #$FF
4F0D   D0 F7      BNE $4F06
```

```
4F0F   20 4F 4F   JSR $4F4F    Verkettung korrigieren
4F12   20 54 52   JSR $5254    PC auf Programmstart setzen

4F15   A5 74      LDA $74      AUTO eingeschaltet ?
4F17   05 75      ORA $75
4F19   F0 31      BEQ $4F4C    Nein, Skip
4F1B   A5 16      LDA $16      AUTO-Offset auf Zeilennummer aufaddieren
4F1D   18         CLC
4F1E   65 74      ADC $74
4F20   85 65      STA $65      und setzen
4F22   A5 17      LDA $17
4F24   65 75      ADC $75
4F26   B0 24      BCS $4F4C    Wenn Wert zu groß, zur Eingabeschleife
4F28   C9 FA      CMP #$FA
4F2A   B0 20      BCS $4F4C    Wenn Wert zu groß, zur Eingabeschleife
4F2C   85 64      STA $64
4F2E   A2 90      LDX #$90     Adresswert in FAC#1
4F30   38         SEC
4F31   20 75 8C   JSR $8C75
4F34   20 42 8E   JSR $8E42    FAC#1 in ASCII ab $0100 wandeln
4F37   A2 00      LDX #$00
4F39    BD 01 01     LDA $0101,X   Zeilennummertext in Tastaturpuffer
eintragen
4F3C   F0 06      BEQ $4F44
4F3E   9D 4A 03   STA $034A,X
4F41   E8         INX
4F42   D0 F5      BNE $4F39
4F44   A9 1D      LDA #$1D     und ein Cursor Right in Tastaturpuffer
4F46   9D 4A 03   STA $034A,X
4F49   E8         INX
4F4A   86 D0      STX $D0      Zeichenanzahl im Tastaturpuffer setzen
4F4C   4C C3 4D   JMP $4DC3    Zur Eingabeschleife

****************************** Verkettung berichtigen

4F4F   A5 2D      LDA $2D      ($24) auf Programmstart
4F51   A4 2E      LDY $2E
4F53   85 24      STA $24
4F55   84 25      STY $25
4F57   18         CLC
4F58   A0 00      LDY #$00
4F5A   20 05 43   JSR $4305    Bank 0 LDA ($24),Y
4F5D   D0 06      BNE $4F65    Wenn Linkadresse gesetzt, Skip
4F5F   C8         INY
4F60   20 05 43   JSR $4305    Programmende erreicht ?
4F63   F0 2D      BEQ $4F92    Ja, Skip
4F65   A0 04      LDY #$04     Zeilentext überlesen
4F67   C8         INY
```

```
4F68   20 05 43   JSR $4305
4F6B   D0 FA      BNE $4F67
4F6D   C8         INY          Verkettung berichtigen
4F6E   98         TYA
4F6F   65 24      ADC $24
4F71   AA         TAX
4F72   A0 00      LDY #$00
4F74   91 24      STA ($24),Y  und setzen
4F76   98         TYA
4F77   65 25      ADC $25
4F79   C8         INY
4F7A   91 24      STA ($24),Y
4F7C   86 24      STX $24      Zeiger auf nächste Zeile setzen
4F7E   85 25      STA $25
4F80   90 D6      BCC $4F58    Unbedingter Sprung

******************************** Programmende setzen

4F82   18         CLC
4F83   A5 24      LDA $24      Zeiger aus der Routine $4F4F
4F85   A4 25      LDY $25
4F87   69 02      ADC #$02     + 2
4F89   90 01      BCC $4F8C
4F8B   C8         INY
4F8C   8D 10 12   STA $1210    ergibt Programmende
4F8F   8C 11 12   STY $1211
4F92   60         RTS

******************************** Zeile in Puffer holen

4F93   A2 00      LDX #$00     Offset auf 0
4F95   20 E5 90   JSR $90E5    BASIN
4F98   C9 0D      CMP #$0D     CR ?
4F9A   D0 03      BNE $4F9F    Nein, Skip
4F9C   4C 8B 55   JMP $558B    Puffer abschließen
4F9F   9D 00 02   STA $0200,X  Zeichen in Puffer
4FA2   E8         INX          Offset erhöhen
4FA3   E0 A1      CPX #$A1     Zuviele Zeichen ?
4FA5   90 EE      BCC $4F95    Nein, weitermachen
4FA7   4C ED A5   JMP $A5ED    'STRING TOO LONG'

******************************** Stapelsuchroutine

4FAA   85 02      STA $02      Suchcode setzen
4FAC   20 47 50   JSR $5047    BASIC-Stackpointer holen
4FAF   A5 3F      LDA $3F      Stapel leer ?
4FB1   C9 FF      CMP #$FF
4FB3   D0 06      BNE $4FBB    Nein, Skip
```

```
4FB5   A5 40      LDA $40
4FB7   C9 09      CMP #$09
4FB9   F0 40      BEQ $4FFB     Ja, Ende
4FBB   8D 03 FF   STA $FF03     Bank 0 vorwählen
4FBE   A0 00      LDY #$00
4FC0   A5 02      LDA $02
4FC2   C9 81      CMP #$81      FOR-Code ?
4FC4   D0 1B      BNE $4FE1
4FC6   D1 3F      CMP ($3F),Y   FOR gefunden ?
4FC8   D0 33      BNE $4FFD     Nein, Ende
4FCA   A0 02      LDY #$02
4FCC   A5 4C      LDA $4C       Variablennamen vergleichen
4FCE   C9 FF      CMP #$FF
4FD0   F0 2B      BEQ $4FFD
4FD2   D1 3F      CMP ($3F),Y
4FD4   D0 07      BNE $4FDD
4FD6   88         DEY
4FD7   A5 4B      LDA $4B
4FD9   D1 3F      CMP ($3F),Y
4FDB   F0 20      BEQ $4FFD     Wenn gleich, gefunden
4FDD   A2 12      LDX #$12
4FDF   D0 0E      BNE $4FEF
4FE1   B1 3F      LDA ($3F),Y   Code gefunden ?
4FE3   C5 02      CMP $02
4FE5   F0 16      BEQ $4FFD     Ja, Ende
4FE7   A2 12      LDX #$12      Offset 18
4FE9   C9 81      CMP #$81      FOR-Code ?
4FEB   F0 02      BEQ $4FEF     Ja, Skip
4FED   A2 05      LDX #$05      Offset 5
4FEF   8A         TXA           Weitersuchen
4FF0   18         CLC
4FF1   65 3F      ADC $3F
4FF3   85 3F      STA $3F
4FF5   90 B8      BCC $4FAF
4FF7   E6 40      INC $40
4FF9   D0 B4      BNE $4FAF
4FFB   A0 01      LDY #$01      Flag für nicht gefunden
4FFD   60         RTS

******************************* BASIC-Stackpointer erhöhen

4FFE   49 FF      EOR #$FF      Zweierkomplement
5000   38         SEC
5001   65 7D      ADC $7D       auf Stapelzeiger aufaddieren
5003   85 7D      STA $7D
5005   A4 7E      LDY $7E
5007   B0 01      BCS $500A
5009   88         DEY
```

```
500A   84 7E      STY $7E
500C   C0 08      CPY #$08    Stapel voll ?
500E   90 34      BCC $5044   Ja, Fehler
5010   D0 04      BNE $5016
5012   C5 7D      CMP $7D
5014   90 2E      BCC $5044   Ja, Fehler
5016   60         RTS

****************************** Schafft Platz im Speicher
                              für Variablen und Programmzeilen

5017   C4 36      CPY $36     Platz schon vorhanden ?
5019   90 28      BCC $5043   Ja, Ende
501B   D0 04      BNE $5021   Nein, Skip
501D   C5 35      CMP $35
501F   90 22      BCC $5043   Ja, Ende
5021   48         PHA         Register retten
5022   A2 09      LDX #$09
5024   98         TYA
5025   48         PHA
5026   B5 59      LDA $59,X
5028   CA         DEX
5029   10 FA      BPL $5025
502B   20 EA 92   JSR $92EA   Garbage-Collection
502e   A2 F7      LDX #$F7    Register holen
5030   68         PLA
5031   95 63      STA $63,X
5033   E8         INX
5034   30 FA      BMI $5030
5036   68         PLA
5037   A8         TAY
5038   68         PLA
5039   C4 36      CPY $36     Platz jetzt vorhanden ?
503B   90 06      BCC $5043   Ja, Ende
503D   D0 05      BNE $5044   Nein, Fehler
503F   C5 35      CMP $35
5041   B0 01      BCS $5044   Nein, Fehler
5043   60         RTS
5044   4C 3A 4D   JMP $4D3A   'OUT OF MEMORY ERROR'

****************************** BASIC-Stackpointer in Hilfszeiger

5047   A5 7D      LDA $7D
5049   85 3F      STA $3F
504B   A5 7E      LDA $7E
504D   85 40      STA $40
504F   60         RTS
```

```
******************************* Hilfszeiger in BASIC-Stackpointer

5050   A5 3F      LDA $3F
5052   85 7D      STA $7D
5054   A5 40      LDA $40
5056   85 7E      STA $7E
5058   60         RTS

******************************* BASIC-Stack um (Y) Bytes leeren

5059   98         TYA
505A   18         CLC
505B   65 7D      ADC $7D
505D   85 7D      STA $7D
505F   90 02      BCC $5063
5061   E6 7E      INC $7E
5063   60         RTS

******************************* Programmzeile suchen

5064   A5 2D      LDA $2D      Zeiger auf Programmanfang
5066   A6 2E      LDX $2E
5068   A0 01      LDY #$01     Offset auf Linkadresse High
506A   85 61      STA $61
506C   86 62      STX $62
506E   20 EC 42   JSR $42EC    Linkadresse High = 0 ?
5071   F0 2B      BEQ $509E    Ja, Ende, nicht gefunden
5073   C8         INY
5074   C8         INY
5075   20 EC 42   JSR $42EC
5078   85 79      STA $79      Zeilennummern High gleich ?
507A   A5 17      LDA $17
507C   C5 79      CMP $79
507E   90 1F      BCC $509F    Kleiner, Ende, nicht gefunden
5080   F0 03      BEQ $5085
5082   88         DEY
5083   D0 0E      BNE $5093
5085   88         DEY
5086   20 EC 42   JSR $42EC
5089   85 79      STA $79      Zeilennummern Low gleich ?
508B   A5 16      LDA $16
508D   C5 79      CMP $79
508F   90 0E      BCC $509F    Kleiner, Ende, nicht gefunden
5091   F0 0C      BEQ $509F    Gleich, gefunden, C-Flag=1
5093   88         DEY
5094   20 EC 42   JSR $42EC    Linkadresse neu setzen
5097   AA         TAX
5098   88         DEY
```

```
5099   20 EC 42   JSR $42EC
509C   B0 CA      BCS $5068    Unbedingter Sprung
509E   18         CLC
509F   60         RTS

****************************** Zeilennummer nach ($16) einlesen

50A0   A2 00      LDX #$00
50A2   86 0A      STX $0A      Zeichenzähler auf 0
50A4   86 16      STX $16      Nummer = 0
50A6   86 17      STX $17
50A8   B0 37      BCS $50E1    Wenn keine Zahl, Ende
50AA   E6 0A      INC $0A      1 Ziffer mehr
50AC   E9 2F      SBC #$2F     ASCII-Wert in Hex-Zahl wandeln
50AE   85 09      STA $09      und setzen
50B0   A5 17      LDA $17
50B2   85 24      STA $24
50B4   C9 19      CMP #$19     Nummer > 6400 ?
50B6   90 03      BCC $50BB    Nein, Skip
50B8   4C 6C 79   JMP $796C    'SYNTAX'
50BB   A5 16      LDA $16      Zahl = Zahl * 10
50BD   0A         ASL
50BE   26 24      ROL $24
50C0   0A         ASL
50C1   26 24      ROL $24
50C3   65 16      ADC $16
50C5   85 16      STA $16
50C7   A5 24      LDA $24
50C9   65 17      ADC $17
50CB   85 17      STA $17
50CD   06 16      ASL $16
50CF   26 17      ROL $17
50D1   A5 16      LDA $16      Auf die Zahl den
50D3   65 09      ADC $09      Ziffernwert aufaddieren
50D5   85 16      STA $16
50D7   90 02      BCC $50DB
50D9   E6 17      INC $17
50DB   20 80 03   JSR $0380    CHRGET nächstes Zeichen
50DE   4C A8 50   JMP $50A8    Zum Schleifenanfang
50E1   60         RTS

****************************** BASIC-Befehl LIST

50E2   20 FB 5E   JSR $5EFB    Zeilenbereich holen
50E5   A0 01      LDY #$01
50E7   20 EC 42   JSR $42EC    Linkadresse = 0 ?
50EA   D0 06      BNE $50F2    Nein, Skip
50EC   88         DEY
```

```
50ED   20 EC 42   JSR $42EC
50F0   F0 2E      BEQ $5120    Ja, Ende
50F2   20 B5 4B   JSR $4BB5    STOP-Taste prüfen
50F5   20 98 55   JSR $5598    (CR) out, neue Zeile
50F8   A0 02      LDY #$02     Offset auf Zeilennummer laden
50FA   20 EC 42   JSR $42EC    Zeilennummer Low-Byte
50FD   AA         TAX          in (X)
50FE   C8         INY
50FF   20 EC 42   JSR $42EC    Zeilennummer  High-Byte in (A)
5102   C5 17      CMP $17      Endnummer überschritten ?
5104   D0 04      BNE $510A
5106   E4 16      CPX $16
5108   F0 02      BEQ $510C    Nein, Skip
510A   B0 14      BCS $5120    Ja, Ende
510C   20 23 51   JSR $5123    LIST eine Zeile
510F   A0 00      LDY #$00     Linkadresse laden
5111   20 EC 42   JSR $42EC
5114   AA         TAX
5115   C8         INY
5116   20 EC 42   JSR $42EC
5119   86 61      STX $61      und als Zeiger auf die nächste
511B   85 62      STA $62      Zeile setzen
511D   4C E5 50   JMP $50E5    Zum Anfang der Schleife
5120   4C 98 55   JMP $5598    (CR) out, neue Zeile

****************************** LIST eine Zeile

5123   A0 03      LDY #$03     Offset auf Programmtext
5125   84 4B      STY $4B      Zeilenzeiger setzen
5127   84 11      STY $11      Hochkommaflag löschen
5129   20 32 8E   JSR $8E32    Zeilennummer ausgeben
512C   A9 20      LDA #$20     Space
512E   A4 4B      LDY $4B      Offset laden
5130   29 7F      AND #$7F     Zeichen anpassen
5132   20 0C 56   JSR $560C    Zeichen ausgeben
5135   C9 22      CMP #$22     '"' ?
5137   D0 06      BNE $513F    Nein, Skip
5139   A5 11      LDA $11      Flag invertieren
513B   49 FF      EOR #$FF
513D   85 11      STA $11
513F   C8         INY          Maximale Zeilenlänge erreicht ?
5140   F0 DE      BEQ $5120    Ja, (CR), Ende
5142   24 55      BIT $55      HELP-Flag gesetzt ?
5144   10 03      BPL $5149    Nein, Skip
5146   20 AC 59   JSR $59AC    HELP Ausgabetest
5149   20 EC 42   JSR $42EC    Zeichen aus Zeile holen
514C   F0 3D      BEQ $518B    Wenn Zeilenende, Ende
514E   6C 06 03   JMP ($0306)  $5151 Umwandlung in Klartext
```

```
5151  10 DF     BPL $5132     Kein Befehl, direkt ausgeben
5153  C9 FF     CMP #$FF      Code für PI ?
5155  F0 DB     BEQ $5132     Ja, direkt ausgeben
5157  24 11     BIT $11       Hochkommaflag gesetzt ?
5159  30 D7     BMI $5132     Ja, Zeichen direkt ausgeben
515B  C9 FE     CMP #$FE      Sondertoken 1 ?
515D  F0 2D     BEQ $518C     Ja, Skip
515F  C9 CE     CMP #$CE      Sondertoken 2 ?
5161  F0 43     BEQ $51A6     Ja, Skip
5163  AA        TAX           Befehlsnummer in (X)
5164  84 4B     STY $4B
5166  A9 44     LDA #$44      Textzeiger auf Befehlstabelle 1 $4417
5168  A0 17     LDY #$17
516A  85 25     STA $25
516C  84 24     STY $24
516E  A0 00     LDY #$00
5170  CA        DEX           Befehl gefunden ?
5171  10 0F     BPL $5182     Ja, Skip
5173  B1 24     LDA ($24),Y   Einen Befehl überlesen
5175  48        PHA
5176  E6 24     INC $24
5178  D0 02     BNE $517C
517A  E6 25     INC $25
517C  68        PLA           Befehlsende erreicht ?
517D  10 F4     BPL $5173     Nein, weiterüberlesen
517F  30 EF     BMI $5170     Nächsten Befehl lesen
5181  C8        INY
5182  B1 24     LDA ($24),Y   Befehl ausgeben
5184  30 A8     BMI $512E     Letztes Zeichen auf Schirm
5186  20 0C 56  JSR $560C     Zeichen ausgeben
5189  D0 F6     BNE $5181     Unbedingter Sprung
518B  60        RTS

518C  AA        TAX           Sondertoken 1
518D  C8        INY
518E  20 EC 42  JSR $42EC     Nächstes Zeichen = 0 ?
5191  F0 9F     BEQ $5132     Ja, direkt ausgeben
5193  84 4B     STY $4B
5195  C9 02     CMP #$02      Token gültig ?
5197  90 27     BCC $51C0     Nein, Skip
5199  C9 27     CMP #$27
519B  B0 23     BCS $51C0     Nein, Skip
519D  69 7E     ADC #$7E      Tokennummern auf $80 anheben
519F  AA        TAX
51A0  A0 09     LDY #$09      Zeiger auf Befehlstabelle 2 $4609
51A2  A9 46     LDA #$46
51A4  D0 C4     BNE $516A     Befehl ausgeben
```

```
51A6   AA         TAX           Sondertoken 2
51A7   C8         INY
51A8   20 EC 42   JSR $42EC     Nächstes Zeichen = 0 ?
51AB   F0 85      BEQ $5132     Ja, direkt ausgeben
51AD   84 4B      STY $4B
51AF   C9 02      CMP #$02      Token gültig ?
51B1   90 0D      BCC $51C0     Nein, Skip
51B3   C9 0B      CMP #$0B
51B5   B0 09      BCS $51C0     Nein, Skip
51B7   69 7E      ADC #$7E      Tokennummern auf $80 anheben
51B9   AA         TAX
51BA   A0 C9      LDY #$C9      Zeiger auf Befehlstabelle 3 $46C9
51BC   A9 46      LDA #$46
51BE   D0 AA      BNE $516A     Befehl ausgeben

51C0   E0 FE      CPX #$FE      Sondertoken 1 ?
51C2   D0 03      BNE $51C7     Nein, Skip
51C4   A2 00      LDX #$00      Flag für Sondertoken 1
51C6   2C         .BYTE $2C
51C7   A2 FF      LDX #$FF      Flag für Sondertoken 2
51C9   38         SEC           Flag für letztes Zeichen
51CA   6C 0E 03   JMP ($030E)
51CD   B0 04      BCS $51D3     Wenn letztes Zeichen, direkt ausgeben
51CF   A0 00      LDY #$00
51D1   F0 AF      BEQ $5182     Befehl ab ($24) ausgeben
51D3   4C 32 51   JMP $5132

******************************** BASIC-Befehl NEW

51D6   F0 01      BEQ $51D9     Bei Trennzeichen, Skip
51D8   60         RTS
51D9   A9 00      LDA #$00      Erste Linkadresse löschen
51DB   A8         TAY
51DC   91 2D      STA ($2D),Y
51DE   C8         INY
51DF   91 2D      STA ($2D),Y
51E1   8D 6F 11   STA $116F     Trace ausschalten
51E4   A5 2D      LDA $2D       Programmendezeiger setzen
51E6   18         CLC
51E7   69 02      ADC #$02
51E9   8D 10 12   STA $1210
51EC   A5 2E      LDA $2E
51EE   69 00      ADC #$00
51F0   8D 11 12   STA $1211
51F3   20 54 52   JSR $5254     PC auf Programmstart
51F6   A9 00      LDA #$00
```

```
****************************** BASIC-Befehl CLR

51F8   D0 55      BNE $524F    Kein Trennzeichen, Ende
51FA   20 7B 92   JSR $927B    CLALL I/O-Kanäle rücksetzen
51FD   A0 00      LDY #$00     DS$ löschen
51FF   84 7A      STY $7A
5201   88         DEY
5202   8C 0C 12   STY $120C    TRAP- und HELP-Flags löschen
5205   8C 09 12   STY $1209
5208   8C 0A 12   STY $120A
520B   8C 08 12   STY $1208
520E   A5 39      LDA $39      Stringstart auf Speicherende
5210   A4 3A      LDY $3A
5212   85 35      STA $35
5214   84 36      STY $36
5216   A9 FF      LDA #$FF     BASIC-Stack initialisieren
5218   A0 09      LDY #$09
521A   85 7D      STA $7D
521C   84 7E      STY $7E
521E   A5 2F      LDA $2F      Variablenstart auf
5220   A4 30      LDY $30      Start der Bank 1
5222   85 31      STA $31
5224   84 32      STY $32
5226   85 33      STA $33
5228   84 34      STY $34
522A   A2 03      LDX #$03     Zeichen für PRINT USING setzen
522C   BD 50 52   LDA $5250,X
522F   9D 04 12   STA $1204,X
5232   CA         DEX
5233   10 F7      BPL $522C
5235   20 E1 5A   JSR $5AE1    RESTORE-Befehl
5238   A2 1B      LDX #$1B     Stringstack initialisieren
523A   86 18      STX $18
523C   68         PLA          Rücksprungadresse retten
523D   A8         TAY
523E   68         PLA
523F   A2 FA      LDX #$FA     Stapelzeiger initialisieren
5241   9A         TXS
5242   48         PHA          Rücksprungadresse wieder setzen
5243   98         TYA
5244   48         PHA
5245   A9 00      LDA #$00
5247   8D 03 12   STA $1203    CONT sperren
524A   85 12      STA $12
524C   8D DF 03   STA $03DF    FAC#1 Überlauf löschen
524F   60         RTS
```

```
****************************** Zeichen für PRINT USING

5250   20 2C 2E 24               ' ' ',' '.' '$'

****************************** PC auf Programmstart

5254   18         CLC
5255   A5 2D      LDA $2D        Programmstart
5257   69 FF      ADC #$FF       minus 1
5259   85 3D      STA $3D        in PC setzen
525B   A5 2E      LDA $2E
525D   69 FF      ADC #$FF
525F   85 3E      STA $3E
5261   60         RTS

****************************** BASIC-Befehl RETURN

5262   68         PLA            Rücksprungadresse löschen
5263   68         PLA
5264   A9 8D      LDA #$8D       Token für GOSUB
5266   20 AA 4F   JSR $4FAA      Token auf BASIC-Stack ?
5269   F0 05      BEQ $5270      Ja, Skip
526B   A2 0C      LDX #$0C       'RETURN WITHOUT GOSUB'
526D   4C 3C 4D   JMP $4D3C
5270   20 50 50   JSR $5050      BASIC-Stackpointer setzen
5273   A0 05      LDY #$05       BASIC-Stack leeren
5275   20 59 50   JSR $5059
5278   88         DEY
5279   B1 3F      LDA ($3F),Y    PC setzen
527B   85 3E      STA $3E
527D   88         DEY
527E   B1 3F      LDA ($3F),Y
5280   85 3D      STA $3D
5282   88         DEY
5283   B1 3F      LDA ($3F),Y    Zeilennummer (und Direktmodusflag)
5285   20 3B A8   JSR $A83B      setzen
5288   B1 3F      LDA ($3F),Y
528A   85 3B      STA $3B
528C   4C 8F 52   JMP $528F      ???

****************************** BASIC-Befehl DATA

528F   20 A2 52   JSR $52A2      Nächstes Statement suchen
5292   98         TYA            PC korrigieren
5293   18         CLC
5294   65 3D      ADC $3D
5296   85 3D      STA $3D
5298   90 02      BCC $529C
```

```
529A   E6 3E      INC $3E
529C   60         RTS

****************************** BASIC-Befehl REM

529D   20 A5 52   JSR $52A5     Bis zum Zeilenende überlesen
52A0   F0 F0      BEQ $5292     Unbedingter Sprung

****************************** Nächstes Trennzeichen suchen

52A2   A2 3A      LDX #$3A      ':' als Trennzeichen vorgeben
52A4   2C         .BYTE $2C
52A5   A2 00      LDX #$00      $00, bis Zeilenende gehen
52A7   86 09      STX $09
52A9   A0 00      LDY #$00      Offset auf 0
52AB   84 0A      STY $0A
52AD   A5 0A      LDA $0A       Zeichen tauschen
52AF   A6 09      LDX $09
52B1   85 09      STA $09
52B3   86 0A      STX $0A
52B5   20 C9 03   JSR $03C9     Zeilenende ?
52B8   F0 E2      BEQ $529C     Ja, Ende
52BA   C5 0A      CMP $0A       Zeichen gefunden ?
52BC   F0 DE      BEQ $529C     Ja, Ende
52BE   C8         INY           Offset erhöhen
52BF   C9 22      CMP #$22      '"' ?
52C1   D0 F2      BNE $52B5     Nein, weiterüberlesen
52C3   F0 E8      BEQ $52AD     Ja, Endeflags vertauschen

****************************** BASIC-Befehl IF

52C5   20 EF 77   JSR $77EF     FRMEVL Ausdruck auswerten
52C8   20 86 03   JSR $0386     CHRGOT
52CB   C9 89      CMP #$89      Token für GOTO ?
52CD   F0 05      BEQ $52D4     Ja, Skip
52CF   A9 A7      LDA #$A7      Token für THEN
52D1   20 5E 79   JSR $795E     Prüft auf Code
52D4   A5 63      LDA $63       Ausdruck ungleich 0 (wahr) ?
52D6   D0 26      BNE $52FE     Ja, Skip
52D8   20 86 03   JSR $0386     CHRGOT
52DB   C9 FE      CMP #$FE      Sondertoken 1 ?
52DD   D0 0B      BNE $52EA     Nein, Skip
52DF   C8         INY
52E0   20 C9 03   JSR $03C9     Nächstes Zeichen
52E3   C9 18      CMP #$18      = Token für BEGIN ?
52E5   D0 03      BNE $52EA     Nein, Skip
52E7   20 20 53   JSR $5320     BEGIN-BEND überlesen
52EA   20 8F 52   JSR $528F     DATA ausführen (Zum nächsten ':')
```

```
52ED   A0 00      LDY #$00
52EF   20 C9 03   JSR $03C9    Momentanes Zeichen = Zeilenende ?
52F2   F0 A9      BEQ $529D    Ja, REM ausführen
52F4   20 80 03   JSR $0380    CHRGET
52F7   C9 D5      CMP #$D5     Token für ELSE ?
52F9   D0 EF      BNE $52EA    Nein, weitertesten
52FB   20 80 03   JSR $0380    CHRGET

52FE   20 86 03   JSR $0386    CHRGOT
5301   F0 17      BEQ $531A    Wenn Trennzeichen, Ende
5303   B0 03      BCS $5308    Wenn keine Zahl, Skip
5305   4C DB 59   JMP $59DB    GOTO
5308   C9 FE      CMP #$FE     Sondertoken 1 ?
530A   D0 0E      BNE $531A    Nein, Skip
530C   C8         INY
530D   20 C9 03   JSR $03C9    Nächstes Zeichen
5310   C9 18      CMP #$18     Token für BEGIN ?
5312   D0 06      BNE $531A    Nein, Skip
5314   20 80 03   JSR $0380    Zwei Zeichen überlesen
5317   20 80 03   JSR $0380
531A   20 86 03   JSR $0386    CHRGOT
531D   4C 3F 4B   JMP $4B3F    Interpreterschleife

******************************* BEGIN-BEND überlesen

5320   20 80 03   JSR $0380    CHRGET
5323   D0 27      BNE $534C    Wenn kein Trennzeichen, Skip
5325   C9 3A      CMP #$3A     ':' ?
5327   F0 F7      BEQ $5320    Ja, weiterlesen
5329   24 7F      BIT $7F      Direktmodus ?
532B   10 4A      BPL $5377    Ja, Fehler
532D   A0 02      LDY #$02
532F   20 C9 03   JSR $03C9    Linkadresse High holen
5332   F0 43      BEQ $5377    Wenn Programmende, Fehler
5334   C8         INY
5335   20 C9 03   JSR $03C9    Zeilennummer setzen
5338   85 3B      STA $3B
533A   C8         INY
533B   20 C9 03   JSR $03C9
533E   85 3C      STA $3C
5340   98         TYA          PC korrigieren
5341   18         CLC
5342   65 3D      ADC $3D
5344   85 3D      STA $3D
5346   90 D8      BCC $5320
5348   E6 3E      INC $3E
534A   D0 D4      BNE $5320    weitertesten
```

```
534C   C9 22      CMP #$22    '"' ?
534E   D0 07      BNE $5357   Nein, Skip
5350   20 7C 53   JSR $537C   Text überlesen
5353   F0 D0      BEQ $5325   Wenn Zeilenende, Skip
5355   D0 C9      BNE $5320   Weitertesten
5357   C9 8F      CMP #$8F    Token für REM ?
5359   D0 06      BNE $5361   Nein, Skip
535B   20 9D 52   JSR $529D   REM ausführen
535E   4C 29 53   JMP $5329   Weitertesten
5361   C9 FE      CMP #$FE    Sondertoken 1 ?
5363   D0 BB      BNE $5320
5365   20 80 03   JSR $0380   CHRGET
5368   C9 19      CMP #$19    Token für BEND ?
536A   F0 0A      BEQ $5376   Ja, Ende
536C   C9 18      CMP #$18    Token für BEGIN
536E   D0 B0      BNE $5320   Nein, weitertesten
5370   20 20 53   JSR $5320   Rekursiver Aufruf
5373   4C 20 53   JMP $5320   Weitertesten
5376   60         RTS
5377   A2 25      LDX #$25    'UNRESOLVED REFERENCE'
5379   4C 3C 4D   JMP $4D3C

****************************** Text in '"' überlesen

537C   A0 00      LDY #$00    Offset auf 0
537E   E6 3D      INC $3D     PC erhöhen
5380   D0 02      BNE $5384
5382   E6 3E      INC $3E
5384   20 C9 03   JSR $03C9   Zeichen aus Programmtext holen
5387   F0 07      BEQ $5390   Wenn Zeilenende, Ende
5389   C9 22      CMP #$22    '"' ?
538B   D0 F1      BNE $537E   Nein, weiterlesen
538D   4C 80 03   JMP $0380   CHRGET
5390   60         RTS

****************************** Falls nötig, BEGIN-BEND überlesen

5391   C9 FE      CMP #$FE    Sondertoken 1 ?
5393   D0 0B      BNE $53A0   Nein, REM-Befehl
5395   C8         INY
5396   20 C9 03   JSR $03C9   Nächstes Zeichen
5399   C9 18      CMP #$18    Token für BEGIN
539B   D0 03      BNE $53A0   Nein, REM-Befehl
539D   20 20 53   JSR $5320   BEGIN-BEND überlesen
53A0   4C 9D 52   JMP $529D   REM-Befehl
```

```
******************************** BASIC-Befehl ON

53A3   20 F4 87   JSR $87F4     Byte-Wert holen
53A6   48         PHA           Token retten
53A7   C9 8D      CMP #$8D      Token für GOSUB ?
53A9   F0 07      BEQ $53B2     Ja, Skip
53AB   C9 89      CMP #$89      Token für GOTO ?
53AD   F0 03      BEQ $53B2     Ja, Skip
53AF   4C 6C 79   JMP $796C     'SYNTAX'
53B2   C6 67      DEC $67       Richtige Nummer erreicht ?
53B4   D0 04      BNE $53BA     Nein, Nummer überlesen
53B6   68         PLA           Befehlstoken holen
53B7   4C 59 4B   JMP $4B59     Befehl ausführen
53BA   20 80 03   JSR $0380     CHRGET
53BD   20 A0 50   JSR $50A0     Zeilennummer lesen
53C0   C9 2C      CMP #$2C      ',' ?
53C2   F0 EE      BEQ $53B2     Ja, weitertesten
53C4   68         PLA           Token vom Stapel löschen
53C5   60         RTS

******************************** BASIC-Befehl LET

53C6   20 AF 7A   JSR $7AAF     Variable suchen
53C9   85 4B      STA $4B       Adresse setzen
53CB   84 4C      STY $4C
53CD   A9 B2      LDA #$B2      Token für '='
53CF   20 5E 79   JSR $795E     Prüft auf Code
53D2   A5 10      LDA $10       Integer-Flag retten
53D4   48         PHA
53D5   A5 0F      LDA $0F       Typ-Flag retten
53D7   48         PHA
53D8   20 EF 77   JSR $77EF     FRMEVL Ausdruck auswerten
53DB   68         PLA
53DC   2A         ROL           Typ-Flag holen
53DD   20 DE 77   JSR $77DE     Typ prüfen
53E0   D0 22      BNE $5404     Wenn String, Skip
53E2   68         PLA           Real ?
53E3   10 15      BPL $53FA     Ja, Skip

******************************** Integer Wertzuweisung

53E5   20 47 8C   JSR $8C47     FAC#1 runden
53E8   20 B4 84   JSR $84B4     FAC#1 in Integer
53EB   A0 00      LDY #$00
53ED   A5 66      LDA $66
53EF   8D 04 FF   STA $FF04     Write in Bank 1 setzen
53F2   91 4B      STA ($4B),Y   Wert in Variable eintragen
53F4   C8         INY
```

```
53F5   A5 67      LDA $67
53F7   91 4B      STA ($4B),Y
53F9   60         RTS

****************************** Real Wertzuweisung

53FA   A6 4B      LDX $4B      Adresse holen
53FC   A4 4C      LDY $4C
53FE   8D 04 FF   STA $FF04    Write in Bank 1 setzen
5401   4C 00 8C   JMP $8C00    Variablenwert eintragen

****************************** String Wertzuweisung

5404   68         PLA
5405   A4 4C      LDY $4C      Variablenadresse High
5407   C0 03      CPY #$03     Variable TI$ ?
5409   D0 72      BNE $547D    Nein, Skip
540B   20 81 87   JSR $8781    FRESTR
540E   C9 06      CMP #$06     Stringlänge = 6 ?
5410   D0 3E      BNE $5450    Nein, Fehler
5412   A0 00      LDY #$00
5414   84 63      STY $63
5416   84 68      STY $68
5418   84 72      STY $72      Stellenzähler löschen
541A   20 48 54   JSR $5448    Prüft nächstes Zeichen auf Ziffer
541D   20 17 8B   JSR $8B17    FAC#1 = FAC#1 * 10
5420   E6 72      INC $72      1 Stelle mehr
5422   A4 72      LDY $72
5424   20 48 54   JSR $5448    Prüft nächstes Zeichen auf Ziffer
5427   20 38 8C   JSR $8C38    FAC#2 = FAC#1
542A   AA         TAX          FAC = 0 ?
542B   F0 05      BEQ $5432    Ja, Skip
542D   E8         INX
542E   8A         TXA
542F   20 22 8B   JSR $8B22    FAC#1 = FAC#1 + FAC#2
5432   A4 72      LDY $72
5434   C8         INY
5435   C0 06      CPY #$06     6 Stellen erreicht ?
5437   D0 DF      BNE $5418    Nein, weitermachen
5439   20 17 8B   JSR $8B17    FAC#1 = FAC#1 * 10
543C   20 C7 8C   JSR $8CC7    FAC#1 in Integerformat
543F   A6 66      LDX $66      Wert holen
5441   A4 65      LDY $65
5443   A5 67      LDA $67
5445   4C DB FF   JMP $FFDB    Uhrzeit setzen
```

```
******************************* Zeichen auf Ziffer prüfen

5448   20 B7 03   JSR $03B7    Zeichen holen
544B   20 90 03   JSR $0390    Zeichen testen
544E   90 03      BCC $5453    Wenn Ziffer, Skip
5450   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'
5453   E9 2F      SBC #$2F     von ASCII zu Zahl
5455   4C B0 8D   JMP $8DB0    in FAC

******************************* String im Programm ? / Zuweisung

5458   68         PLA
5459   C8         INY
545A   C5 36      CMP $36      String im Programmtext ?
545C   90 18      BCC $5476    Ja, Skip
545E   D0 08      BNE $5468
5460   88         DEY
5461   20 E7 42   JSR $42E7
5464   C5 35      CMP $35
5466   90 0E      BCC $5476    Ja, Skip
5468   A4 67      LDY $67
546A   C4 30      CPY $30
546C   90 08      BCC $5476    Ja, Skip
546E   D0 24      BNE $5494    Nein, weiter
5470   A5 66      LDA $66
5472   C5 2F      CMP $2F
5474   B0 1E      BCS $5494    Nein, weiter
5476   A5 66      LDA $66
5478   A4 67      LDY $67
547A   4C B2 54   JMP $54B2

******************************* Wertzuweisung an normalen String

547D   A0 02      LDY #$02
547F   20 E7 42   JSR $42E7    Stringadresse High
5482   C5 7C      CMP $7C      = DS$ Adresse High ?
5484   D0 D4      BNE $545A    Nein, Skip
5486   48         PHA          Retten
5487   88         DEY
5488   20 E7 42   JSR $42E7    Stringadresse Low
548B   C5 7B      CMP $7B      = DS$ Adresse Low ?
548D   D0 C9      BNE $5458    Nein, Skip
548F   A5 7A      LDA $7A      DS$ = "" ?
5491   F0 C5      BEQ $5458    Ja, Skip
5493   68         PLA          Adresse High holen
5494   A0 00      LDY #$00
5496   20 E7 42   JSR $42E7    Stringlänge holen
5499   20 88 86   JSR $8688    Speicherplatz prüfen
```

```
549C   A5 52      LDA $52       String in Stringbereich übertragen
549E   A4 53      LDY $53
54A0   85 70      STA $70
54A2   84 71      STY $71
54A4   20 4E 87   JSR $874E     String in reservierten Bereich übertragen
54A7   A5 70      LDA $70
54A9   A4 71      LDY $71
54AB   20 E0 87   JSR $87E0     Stringzeiger von Stringstack löschen
54AE   A9 63      LDA #$63
54B0   A0 00      LDY #$00
54B2   85 52      STA $52
54B4   84 53      STY $53
54B6   85 24      STA $24
54B8   84 25      STY $25
54BA   20 E0 87   JSR $87E0     Stringzeiger löschen
54BD   20 F6 54   JSR $54F6     String im Stringbereich
54C0   90 0E      BCC $54D0     Nein, Skip
54C2   A0 00      LDY #$00
54C4   A5 4B      LDA $4B       Trailer auf Deskriptor setzen
54C6   8D 04 FF   STA $FF04     Write in Bank 1 setzen
54C9   91 24      STA ($24),Y
54CB   C8         INY
54CC   A5 4C      LDA $4C
54CE   91 24      STA ($24),Y
54D0   A5 4B      LDA $4B       Zeiger auf Deskriptor neu setzen
54D2   85 24      STA $24
54D4   A5 4C      LDA $4C
54D6   85 25      STA $25
54D8   20 F6 54   JSR $54F6     Alter String im Stringbereich ?
54DB   90 0C      BCC $54E9     Nein, Skip
54DD   88         DEY           Deskriptor löschen
54DE   A9 FF      LDA #$FF
54E0   8D 04 FF   STA $FF04     Write in Bank 1 setzen
54E3   91 24      STA ($24),Y   String für ungültig erklären
54E5   88         DEY
54E6   8A         TXA
54E7   91 24      STA ($24),Y
54E9   A0 02      LDY #$02      Stringparameter in Variable übertragen
54EB   A9 52      LDA #$52
54ED   20 AB 03   JSR $03AB
54F0   91 4B      STA ($4B),Y
54F2   88         DEY
54F3   10 F6      BPL $54EB
54F5   60         RTS

****************************** Test auf DS$ oder String im Programmtext

54F6   A0 00      LDY #$00
```

```
54F8   20 B7 03   JSR $03B7    Länge holen
54FB   48         PHA          und retten
54FC   F0 39      BEQ $5537    Wenn Länge = 0, Ende
54FE   C8         INY
54FF   20 B7 03   JSR $03B7    Adresse laden
5502   AA         TAX
5503   C8         INY
5504   20 B7 03   JSR $03B7
5507   C5 3A      CMP $3A      Adresse oberhalb Stringspeicher ?
5509   90 06      BCC $5511
550B   D0 2A      BNE $5537    Ja, Ende
550D   E4 39      CPX $39
550F   B0 26      BCS $5537    Ja, Ende
5511   20 B7 03   JSR $03B7
5514   C5 36      CMP $36      Adresse unterhalb Stringspeicher ?
5516   90 1F      BCC $5537    Ja, Ende
5518   D0 04      BNE $551E
551A   E4 35      CPX $35
551C   90 19      BCC $5537    Ja, Ende
551E   C5 7C      CMP $7C      Deskriptor für DS$ ?
5520   D0 04      BNE $5526
5522   E4 7B      CPX $7B
5524   F0 11      BEQ $5537    Ja, Ende
5526   86 24      STX $24      Adresse setzen
5528   85 25      STA $25
552A   68         PLA
552B   AA         TAX          Länge in (X)
552C   18         CLC          Zeiger auf Ende setzen
552D   65 24      ADC $24
552F   85 24      STA $24
5531   90 02      BCC $5535
5533   E6 25      INC $25
5535   38         SEC          Flag für String im Stringbereich
5536   60         RTS
5537   68         PLA
5538   18         CLC          Flag für String nicht im Stringbereich
5539   60         RTS

******************************** BASIC-Befehl PRINT#

553A   20 40 55   JSR $5540    CMD-Befehl
553D   4C 58 56   JMP $5658    und CLRCH

******************************** BASIC-Befehl CMD

5540   20 F4 87   JSR $87F4    Holt Byte-Wert
5543   F0 05      BEQ $554A    Kein Zeichen mehr, Skip
5545   A9 2C      LDA #$2C     ','
```

```
5547   20 5E 79   JSR $795E     Prüft auf Code
554A   08         PHP
554B   86 15      STX $15       Nummer des Ausgabegeräts setzen
554D   20 EB 90   JSR $90EB     CKOUT Ausgabe setzen
5550   28         PLP
5551   4C 5A 55   JMP $555A     Zum PRINT-Befehl

5554   20 E5 55   JSR $55E5     String ausgeben
5557   20 86 03   JSR $0386     CHRGOT

****************************** BASIC-Befehl PRINT

555A   F0 3C      BEQ $5598     Wenn Trennzeichen, CR out
555C   C9 FB      CMP #$FB      Token für USING ?
555E   D0 03      BNE $5563     Nein, Skip
5560   4C 20 95   JMP $9520     PRINT USING-Befehl
5563   F0 43      BEQ $55A8     Wenn Trennzeichen, Ende
5565   C9 A3      CMP #$A3      Token für TAB( ?
5567   F0 50      BEQ $55B9     Ja, Skip
5569   C9 A6      CMP #$A6      Token für SPC( ?
556B   18         CLC           Flag für SPC( setzen
556C   F0 4B      BEQ $55B9     Ja, zu SPC(
556E   C9 2C      CMP #$2C      ',' ?
5570   F0 37      BEQ $55A9     Ja, Skip
5572   C9 3B      CMP #$3B      ';' ?
5574   F0 5E      BEQ $55D4     Ja, Skip
5576   20 EF 77   JSR $77EF     FRMEVL Ausdruck auswerten
5579   24 0F      BIT $0F       Typ = String ?
557B   30 D7      BMI $5554     Ja, ausgeben
557D   20 42 8E   JSR $8E42     FAC#1 in String umwandeln
5580   20 9A 86   JSR $869A     Parameter holen
5583   20 E5 55   JSR $55E5     String ausgeben
5586   20 00 56   JSR $5600     CURSOR-RIGHT oder SPACE ausgeben
5589   D0 CC      BNE $5557     Weitermachen

558B   A9 00      LDA #$00      Eingabepuffer
558D   9D 00 02   STA $0200,X   mit $00 abschließen
5590   A2 FF      LDX #$FF      Zeiger auf Puffer setzen
5592   A0 01      LDY #$01
5594   A5 15      LDA $15       Ausgabegerät = Tastatur ?
5596   D0 10      BNE $55A8     Nein, Skip
5598   A9 0D      LDA #$0D      CR out
559A   20 0C 56   JSR $560C
559D   24 15      BIT $15       Filenummer > 128 ?
559F   10 05      BPL $55A6     Nein, Skip
55A1   A9 0A      LDA #$0A      LF out
55A3   20 0C 56   JSR $560C
```

```
55A6   49 FF      EOR #$FF
55A8   60         RTS

55A9   38         SEC          Zehnertabulator
55AA   20 8D 92   JSR $928D    CURSOR-Position holen
55AD   98         TYA
55AE   38         SEC
55AF   E9 0A      SBC #$0A     - 10
55B1   B0 FC      BCS $55AF    Wenn positiv, Skip
55B3   49 FF      EOR #$FF     Zweierkomplement bilden
55B5   69 01      ADC #$01
55B7   D0 16      BNE $55CF

****************************** TAB( und SPC(

55B9   08         PHP
55BA   38         SEC
55BB   20 8D 92   JSR $928D    CURSOR-Position holen
55BE   84 0B      STY $0B      und retten
55C0   20 F1 87   JSR $87F1    Byte-Wert holen
55C3   C9 29      CMP #$29     ')' ?
55C5   D0 13      BNE $55DA    Nein, Fehler
55C7   28         PLP          SPC( ?
55C8   90 06      BCC $55D0    Ja, Skip
55CA   8A         TXA
55CB   E5 0B      SBC $0B      TAB-Wert > CURSOR-Position ?
55CD   90 05      BCC $55D4    Nein, Ende
55CF   AA         TAX
55D0   E8         INX
55D1   CA         DEX
55D2   D0 09      BNE $55DD
55D4   20 80 03   JSR $0380    CHRGET
55D7   4C 63 55   JMP $5563    Weitermachen
55DA   4C 6C 79   JMP $796C

55DD   20 00 56   JSR $5600    CURSOR-RIGHT oder SPACE ausgeben
55E0   D0 EF      BNE $55D1    Weitermachen

****************************** String ausgeben

55E2   20 9A 86   JSR $869A    Parameter holen
55E5   20 81 87   JSR $8781    FRESTR
55E8   AA         TAX
55E9   A0 00      LDY #$00
55EB   E8         INX
55EC   CA         DEX          Stringende ?
55ED   F0 B9      BEQ $55A8    Ja, Ende
55EF   20 B7 03   JSR $03B7    Zeichen aus String holen
```

```
55F2   20 0C 56   JSR $560C    und ausgeben
55F5   C8         INY
55F6   C9 0D      CMP #$0D     CR erreicht ?
55F8   D0 F2      BNE $55EC    Nein, weitermachen
55FA   20 A6 55   JSR $55A6    Fehler ! Richtig : JSR $559D
55FD   4C EC 55   JMP $55EC    Weitermachen

****************************** CURSOR-RIGHT oder SPACE ausgeben

5600   A5 15      LDA $15      Ausgabe in Bildschirm ?
5602   F0 03      BEQ $5607    Ja, CURSOR-RIGHT out
5604   A9 20      LDA #$20     SPACE out
5606   2C         .BYTE $2C
5607   A9 1D      LDA #$1D     CURSOR-RIGHT out
5609   2C         .BYTE $2C
560A   A9 3F      LDA #$3F     '?' out
560C   20 DF 90   JSR $90DF
560F   29 FF      AND #$FF     Flags setzen
5611   60         RTS

****************************** BASIC-Befehl GET

5612   20 D9 84   JSR $84D9    Test auf Direktmodus
5615   85 77      STA $77      Zeichen retten
5617   C9 23      CMP #$23     '#' ?
5619   F0 0A      BEQ $5625    Ja, GET#
561B   C9 F9      CMP #$F9     Token für KEY ?
561D   D0 16      BNE $5635    Nein, Skip
561F   20 80 03   JSR $0380    CHRGET
5622   4C 35 56   JMP $5635    GETKEY
5625   20 80 03   JSR $0380    CHRGET
5628   20 F4 87   JSR $87F4    Byte-Wert holen
562B   A9 2C      LDA #$2C     ','
562D   20 5E 79   JSR $795E    Prüft auf Code
5630   86 15      STX $15      Eingabegerät setzen
5632   20 FD 90   JSR $90FD    CHKIN Eingabe setzen
5635   A2 01      LDX #$01     Zeiger auf Pufferende $0201
5637   A0 02      LDY #$02
5639   A9 00      LDA #$00     Puffer mit 0 abschließen
563B   8D 01 02   STA $0201
563E   A9 40      LDA #$40     Flag für GET
5640   20 B2 56   JSR $56B2    Wertzuweisung an Variable
5643   A6 15      LDX $15      Eingabegerät = Tastatur ?
5645   D0 13      BNE $565A    Nein, CLRCH
5647   60         RTS
```

```
****************************** BASIC-Befehl INPUT#

5648   20 F4 87   JSR $87F4    Byte-Wert holen
564B   A9 2C      LDA #$2C     ','
564D   20 5E 79   JSR $795E    Prüft auf Code
5650   86 15      STX $15      Eingabegerät setzen
5652   20 FD 90   JSR $90FD    CHKIN
5655   20 71 56   JSR $5671    INPUT ohne Dialogstring
5658   A5 15      LDA $15
565A   20 6F 92   JSR $926F    CLRCH
565D   A2 00      LDX #$00     Eingabe auf Tastatur
565F   86 15      STX $15
5661   60         RTS

****************************** BASIC-Befehl INPUT

5662   C9 22      CMP #$22     '"' ?
5664   D0 0B      BNE $5671    Nein, Skip
5666   20 13 79   JSR $7913    Dialogstring holen
5669   A9 3B      LDA #$3B     ';'
566B   20 5E 79   JSR $795E    Prüft auf Code
566E   20 E5 55   JSR $55E5    String ausgeben
5671   20 D9 84   JSR $84D9    Test auf Direktmodus
5674   A9 2C      LDA #$2C     ','
5676   8D FF 01   STA $01FF    als erstes Zeichen in Puffer
5679   20 9C 56   JSR $569C    Eingabezeile holen
567C   A5 15      LDA $15      Eingabegerät = Tastatur ?
567E   F0 0D      BEQ $568D    Ja, Skip
5680   20 51 92   JSR $9251    Status holen
5683   29 02      AND #$02     Time-out ?
5685   F0 06      BEQ $568D    Nein, Skip
5687   20 58 56   JSR $5658    CLRCH, Eingabe von Tastatur
568A   4C 8F 52   JMP $528F    Nächstes Statement aufsuchen
568D   AD 00 02   LDA $0200    Zeichen im Puffer ?
5690   D0 1E      BNE $56B0    Ja, Skip
5692   A5 15      LDA $15      Eingabe von Tastatur ?
5694   D0 E3      BNE $5679    Nein, Ende
5696   20 A2 52   JSR $52A2    PC auf nächstes Statement
5699   4C 92 52   JMP $5292

569C   A5 15      LDA $15      Eingabe von Tastatur ?
569E   D0 06      BNE $56A6    Nein, Skip
56A0   20 0A 56   JSR $560A    '?' ausgeben
56A3   20 04 56   JSR $5604    SPACE ausgeben
56A6   4C 93 4F   JMP $4F93    Zeile in Eingabepuffer holen
```

```
****************************** BASIC-Befehl READ

56A9   A6 43      LDX $43      DATA-Zeiger holen
56AB   A4 44      LDY $44
56AD   A9 98      LDA #$98     Flag für READ setzen
56AF   2C         .BYTE $2C
56B0   A9 00      LDA #$00     Flag für INPUT oder INPUT#
56B2   85 13      STA $13      setzen
56B4   86 45      STX $45      Eingabezeiger setzen
56B6   84 46      STY $46
56B8   20 AF 7A   JSR $7AAF    Variable suchen
56BB   85 4B      STA $4B      Adresse setzen
56BD   84 4C      STY $4C
56BF   A2 01      LDX #$01
56C1   B5 3D      LDA $3D,X
56C3   95 4D      STA $4D,X    PC retten
56C5   B5 45      LDA $45,X    PC auf Eingabezeiger
56C7   95 3D      STA $3D,X
56C9   CA         DEX
56CA   10 F5      BPL $56C1
56CC   20 86 03   JSR $0386    CHRGOT
56CF   D0 31      BNE $5702
56D1   24 13      BIT $13      GET ?
56D3   50 1A      BVC $56EF    Nein, Skip
56D5   A5 77      LDA $77
56D7   C9 F9      CMP #$F9     GETKEY ?
56D9   D0 08      BNE $56E3    Nein, Skip
56DB   20 09 91   JSR $9109    Warten auf ein Zeichen
56DE   AA         TAX
56DF   F0 FA      BEQ $56DB
56E1   D0 03      BNE $56E6    Unbedingter Sprung
56E3   20 09 91   JSR $9109    GETIN
56E6   8D 00 02   STA $0200    Zeichen in Puffer
56E9   A2 FF      LDX #$FF     Zeiger auf Puffer setzen
56EB   A0 01      LDY #$01
56ED   D0 0F      BNE $56FE    weiter

56EF   10 03      BPL $56F4    Wenn kein READ, Skip
56F1   4C CA 57   JMP $57CA
56F4   A5 15      LDA $15      Eingabe von Tastatur ?
56F6   D0 03      BNE $56FB    Nein, Skip
56F8   20 0A 56   JSR $560A    '?' ausgeben
56FB   20 9C 56   JSR $569C    Eingabezeile holen
56FE   86 3D      STX $3D      PC setzen
5700   84 3E      STY $3E
5702   20 80 03   JSR $0380    CHRGET
5705   24 0F      BIT $0F      Typ = String ?
5707   10 31      BPL $573A    Nein, Skip
```

```
5709   24 13      BIT $13      GET ?
570B   50 09      BVC $5716    Nein, Skip
570D   E8         INX
570E   86 3D      STX $3D
5710   A9 00      LDA #$00
5712   85 09      STA $09
5714   F0 0C      BEQ $5722    Unbedingter Sprung
5716   85 09      STA $09
5718   C9 22      CMP #$22     '"' ?
571A   F0 07      BEQ $5723    Ja, Skip
571C   A9 3A      LDA #$3A     ':'
571E   85 09      STA $09
5720   A9 2C      LDA #$2C     ','
5722   18         CLC
5723   85 0A      STA $0A
5725   A5 3D      LDA $3D      PC laden
5727   A4 3E      LDY $3E
5729   69 00      ADC #$00     und 1 addieren
572B   90 01      BCC $572E
572D   C8         INY
572E   20 A0 86   JSR $86A0    String übernehmen
5731   20 1F 79   JSR $791F    PC hinter String setzen
5734   20 05 54   JSR $5405    String an Variable zuweisen
5737   4C 44 57   JMP $5744
573A   A2 00      LDX #$00     String in Zahl umwandeln
573C   20 22 8D   JSR $8D22
573F   A5 10      LDA $10
5741   20 E3 53   JSR $53E3    FAC#1 in Variable übertragen
5744   20 86 03   JSR $0386    CHRGET
5747   F0 3B      BEQ $5784    Wenn Ende, Skip
5749   C9 2C      CMP #$2C     ',' ?
574B   F0 37      BEQ $5784    Ja, Skip
574D   A5 13      LDA $13
574F   F0 0A      BEQ $575B    Wenn INPUT, Skip
5751   30 04      BMI $5757    Wenn READ, Skip
5753   A6 15      LDX $15      Eingabe von Tastatur ?
5755   D0 08      BNE $575F    Nein, Skip
5757   A2 16      LDX #$16     'TYPE MISMATCH'
5759   D0 06      BNE $5761
575B   A5 15      LDA $15      Eingabe von Tastatur ?
575D   F0 05      BEQ $5764    Ja, Skip
575F   A2 18      LDX #$18     'FILE DATA'
5761   4C 3C 4D   JMP $4D3C
5764   20 81 92   JSR $9281    String ausgeben

******************************* Textkonstante '?REDO FROM START'

5767   3F 52 45 44 4F 20 46 52 '?REDO FROM START' (CR)
```

```
576F   4F 4D 20 53 54 41 52 54
5777   0D 00

****************************** PC auf Befehlsanfang setzen

5779   AD 02 12   LDA $1202    PC auf Befehlsanfang setzen
577C   AC 03 12   LDY $1203
577F   85 3D      STA $3D
5781   84 3E      STY $3E
5783   60         RTS

******************************

5784   A2 01      LDX #$01
5786   B5 3D      LDA $3D,X
5788   95 45      STA $45,X    PC in Eingabezeiger
578A   B5 4D      LDA $4D,X    PC wieder setzen
578C   95 3D      STA $3D,X
578E   CA         DEX
578F   10 F5      BPL $5786
5791   20 86 03   JSR $0386    CHRGOT
5794   F0 06      BEQ $579C    Wenn Trennzeichen, Skip
5796   20 5C 79   JSR $795C    Test auf Komma
5799   4C B8 56   JMP $56B8    weitermachen
579C   A5 45      LDA $45      Variablenzeiger
579E   A4 46      LDY $46
57A0   A6 13      LDX $13      Readbefehl ?
57A2   10 05      BPL $57A9    Nein, Skip
57A4   85 43      STA $43      in DATA-Zeiger setzen
57A6   84 44      STY $44
57A8   60         RTS
57A9   A0 00      LDY #$00     Falls nötig
57AB   A9 45      LDA #$45
57AD   20 9F 03   JSR $039F    Noch Zeichen in Eingabe vorhanden
57B0   F0 17      BEQ $57C9    Nein, Ende
57B2   A5 15      LDA $15      Eingabe von Tastatur ?
57B4   D0 13      BNE $57C9    Nein, Skip
57B6   20 81 92   JSR $9281    '?EXTRA IGNORED'

****************************** Textkonstante '?EXTRA IGNORED'

57B9   3F 45 58 54 52 41 20 49 '?EXTRA IGNORED' (CR)
57C1   47 4E 4F 52 45 44 0D 00

******************************

57C9   60         RTS
```

```
****************************** Nächstes DATA suchen

57CA   20 A2 52   JSR $52A2    Nächstes Trennzeichen suchen
57CD   C8         INY
57CE   AA         TAX          Zeilenende ?
57CF   D0 15      BNE $57E6    Nein, Skip
57D1   A2 0D      LDX #$0D     Nummer für 'OUT OF DATA'
57D3   C8         INY
57D4   20 C9 03   JSR $03C9    Programmende ?
57D7   F0 40      BEQ $5819    Ja, Fehler
57D9   C8         INY
57DA   20 C9 03   JSR $03C9    DATA-Zeiger setzen
57DD   85 41      STA $41
57DF   C8         INY
57E0   20 C9 03   JSR $03C9
57E3   C8         INY
57E4   85 42      STA $42
57E6   20 92 52   JSR $5292    PC auf nächstes Statement
57E9   20 86 03   JSR $0386    CHRGOT
57EC   AA         TAX
57ED   E0 83      CPX #$83     Token für DATA ?
57EF   D0 D9      BNE $57CA    Nein, weitersuchen
57F1   4C 02 57   JMP $5702    Daten lesen

****************************** BASIC-Befehl NEXT

57F4   D0 13      BNE $5809    Wenn Variable angegeben, Skip
57F6   A0 FF      LDY #$FF     Flag für keine Variable
57F8   D0 14      BNE $580E    Unbedingter Sprung
57FA   A0 12      LDY #$12
57FC   20 59 50   JSR $5059    BASIC-Stack um (Y) Bytes leeren
57FF   20 86 03   JSR $0386    CHRGOT
5802   C9 2C      CMP #$2C     Zeichen = ',' ?
5804   D0 71      BNE $5877    Nein, Ende
5806   20 80 03   JSR $0380    CHRGET
5809   20 AF 7A   JSR $7AAF    Variable suchen
580C   85 4B      STA $4B      Adresse setzen
580E   84 4C      STY $4C
5810   A9 81      LDA #$81     Token für FOR
5812   20 AA 4F   JSR $4FAA    Im BASIC-Stack ?
5815   F0 05      BEQ $581C    Ja, Skip
5817   A2 0A      LDX #$0A     'NEXT WITHOUT FOR'
5819   4C 3C 4D   JMP $4D3C
581C   20 50 50   JSR $5050    BASIC-Stackpointer setzen
581F   A5 3F      LDA $3F      Zeiger auf Schleifenwert
5821   18         CLC          im BASIC-Stack setzen
5822   69 03      ADC #$03
5824   A4 40      LDY $40
```

```
5826   90 01      BCC $5829
5828   C8         INY
5829   20 D4 8B   JSR $8BD4    Variable von BASIC-Stack in FAC#1
582C   A0 08      LDY #$08
582E   B1 3F      LDA ($3F),Y
5830   85 68      STA $68
5832   A0 01      LDY #$01
5834   B1 3F      LDA ($3F),Y
5836   48         PHA
5837   AA         TAX
5838   C8         INY
5839   B1 3F      LDA ($3F),Y
583B   48         PHA
583C   A8         TAY
583D   8A         TXA
583E   20 45 88   JSR $8845    FAC#1=Konstante (A)/(Y) + FAC#1
5841   68         PLA
5842   A8         TAY
5843   68         PLA
5844   AA         TAX
5845   8D 04 FF   STA $FF04    Write in Bank 1 setzen
5848   20 00 8C   JSR $8C00    FAC#1 nach (X)/(Y) mit Rundung
584B   A5 3F      LDA $3F
584D   18         CLC
584E   69 09      ADC #$09
5850   A4 40      LDY $40
5852   90 01      BCC $5855
5854   C8         INY
5855   8D 03 FF   STA $FF03    Read aus Bank 0 setzen
5858   20 87 8C   JSR $8C87    Vergleich Konstante (A)/(Y) mit FAC#1
585B   A0 08      LDY #$08
585D   38         SEC
585E   F1 3F      SBC ($3F),Y
5860   F0 98      BEQ $57FA
5862   A0 11      LDY #$11
5864   B1 3F      LDA ($3F),Y
5866   85 3D      STA $3D
5868   88         DEY
5869   B1 3F      LDA ($3F),Y
586B   85 3E      STA $3E
586D   88         DEY
586E   B1 3F      LDA ($3F),Y
5870   85 3C      STA $3C
5872   88         DEY
5873   B1 3F      LDA ($3F),Y
5875   85 3B      STA $3B
5877   60         RTS
```

```
******************************

5878   20 5C 79   JSR $795C    Test auf Komma

****************************** BASIC-Befehl DIM

587B   AA         TAX
587C   20 B4 7A   JSR $7AB4    Variable dimensionieren
587F   20 86 03   JSR $0386    CHRGET
5882   D0 F4      BNE $5878    Wenn kein Trennzeichen, weitermachen
5884   60         RTS

****************************** BASIC-Befehl SYS

5885   20 12 88   JSR $8812    Adresse holen
5888   A5 16      LDA $16      Adresse setzen
588A   85 04      STA $04
588C   A5 17      LDA $17
588E   85 03      STA $03
5890   AD D5 03   LDA $03D5    Bank setzen
5893   85 02      STA $02
5895   20 1E 9E   JSR $9E1E    Falls nötig
5898   90 02      BCC $589C
589A   86 06      STX $06      (A) setzen
589C   20 1E 9E   JSR $9E1E    Falls nötig
589F   90 02      BCC $58A3
58A1   86 07      STX $07      (X) setzen
58A3   20 1E 9E   JSR $9E1E    Falls nötig
58A6   90 02      BCC $58AA
58A8   86 08      STX $08      (Y) setzen
58AA   20 1E 9E   JSR $9E1E    Falls nötig
58AD   90 02      BCC $58B1
58AF   86 05      STX $05      Status setzen
58B1   4C 6E FF   JMP $FF6E    JSRFAR Programmaufruf

****************************** BASIC-Befehle TRON/TROFF

58B4   A9 FF      LDA #$FF     TRON
58B6   2C         .BYTE $2C

58B7   A9 00      LDA #$00     TROFF
58B9   8D 6F 11   STA $116F    Traceflag setzen
58BC   60         RTS

****************************** BASIC-Befehl RREG

58BD   A9 00      LDA #$00     Offset auf 0
58BF   85 0D      STA $0D
```

```
58C1   20 86 03   JSR $0386     CHRGOT
58C4   F0 37      BEQ $58FD
58C6   C9 2C      CMP #$2C      ',' ?
58C8   F0 21      BEQ $58EB     Ja, Skip
58CA   20 AF 7A   JSR $7AAF     Variable suchen
58CD   85 4B      STA $4B       Adresse setzen
58CF   84 4C      STY $4C
58D1   A5 0F      LDA $0F       String ?
58D3   D0 29      BNE $58FE     Ja, Fehler
58D5   A4 0D      LDY $0D       Offset laden
58D7   B9 06 00   LDA $0006,Y   Registerwert laden
58DA   C0 03      CPY #$03      Prozessorstatus laden ?
58DC   D0 02      BNE $58E0     Nein, Skip
58DE   A5 05      LDA $05       Prozessorstatus laden
58E0   A8         TAY           Wert als Low-Byte laden
58E1   A9 00      LDA #$00      High-Byte auf 0
58E3   20 3C 79   JSR $793C     Integerwert in FAC#1
58E6   A5 10      LDA $10       Datentyp laden
58E8   20 E3 53   JSR $53E3     Wertzuweisung
58EB   E6 0D      INC $0D       Offset erhöhen
58ED   A5 0D      LDA $0D       Alle Werte übernommen ?
58EF   C9 04      CMP #$04
58F1   B0 0A      BCS $58FD     Ja, Ende
58F3   20 86 03   JSR $0386     Ende erreicht ?
58F6   F0 05      BEQ $58FD     Ja, Ende
58F8   20 80 03   JSR $0380     Falls nötig
58FB   D0 C4      BNE $58C1     weitermachen
58FD   60         RTS
58FE   4C E7 77   JMP $77E7     'TYPE MISMATCH'

******************************** BASIC-Befehl MID$

5901   20 59 79   JSR $7959     Test auf '('
5904   20 AF 7A   JSR $7AAF     Variable suchen
5907   85 4B      STA $4B       Adresse setzen
5909   84 4C      STY $4C
590B   20 DD 77   JSR $77DD     Test auf String
590E   20 09 88   JSR $8809     Byte-Wert holen
5911   CA         DEX
5912   86 78      STX $78       Einsetzposition setzen
5914   C9 29      CMP #$29      ')' ?
5916   F0 04      BEQ $591C     Ja, Skip
5918   20 09 88   JSR $8809     Byte-Wert holen
591B   2C         .BYTE $2C
591C   A2 FF      LDX #$FF
591E   86 77      STX $77       Übernahmelänge setzen
5920   20 56 79   JSR $7956
5923   A9 B2      LDA #$B2      Token für '='
```

```
5925   20 5E 79   JSR $795E     Prüft auf Code
5928   20 EF 77   JSR $77EF     Ausdruck auswerten
592B   20 DD 77   JSR $77DD     und auf String prüfen
592E   A0 02      LDY #$02
5930   A9 4B      LDA #$4B
5932   20 AB 03   JSR $03AB
5935   99 5D 00   STA $005D,Y   Deskriptor des alten Strings setzen
5938   20 E7 42   JSR $42E7
593B   99 60 00   STA $0060,Y   Deskriptor des Einsetzstrings setzen
593E   88         DEY
593F   10 EF      BPL $5930
5941   38         SEC
5942   A5 61      LDA $61
5944   E5 78      SBC $78
5946   85 61      STA $61
5948   B0 02      BCS $594C
594A   C6 62      DEC $62
594C   A5 77      LDA $77       Übernahmelänge kleiner als
594E   C5 60      CMP $60       Länge des Einsetzstrings ?
5950   90 02      BCC $5954     Ja, Fehler
5952   A5 60      LDA $60       Länge des Einsetzstrings = 0 ?
5954   AA         TAX
5955   F0 18      BEQ $596F     Ja, Ende
5957   18         CLC           + Einsetzposition
5958   65 78      ADC $78       zu groß ?
595A   B0 16      BCS $5972     Ja, Fehler
595C   C5 5D      CMP $5D       größer als Länge des alten Strings ?
595E   90 02      BCC $5962
5960   D0 10      BNE $5972     Ja, Fehler
5962   A4 78      LDY $78       Einsetzposition laden
5964   A9 61      LDA #$61
5966   20 AB 03   JSR $03AB     Einsetzstring byteweise
5969   91 5E      STA ($5E),Y   in alten String übertragen
596B   C8         INY
596C   CA         DEX
596D   D0 F5      BNE $5964
596F   4C 81 87   JMP $8781     FRESTR
5972   4C 28 7D   JMP $7D28     'ILLEGAL QUANTITY'

****************************** BASIC-Befehl AUTO

5975   20 F0 84   JSR $84F0     Test auf Programmmodus
5978   20 A0 50   JSR $50A0     Zeilennummer lesen
597B   A5 16      LDA $16       AUTO-Offset setzen
597D   85 74      STA $74
597F   A5 17      LDA $17
5981   85 75      STA $75
```

```
5983   4C 37 4D   JMP $4D37    'READY.'

****************************** BASIC-Befehl HELP

5986   AE 08 12   LDX $1208    Fehler aufgetreten
5989   E8         INX
598A   F0 1D      BEQ $59A9    Nein, Ende
598C   AD 09 12   LDA $1209    Zeilennummer setzen
598F   AC 0A 12   LDY $120A
5992   85 16      STA $16
5994   84 17      STY $17
5996   20 64 50   JSR $5064    Zeile suchen
5999   90 0E      BCC $59A9    Wenn nicht gefunden, Ende
599B   66 55      ROR $55      In HELP-Flag BIT 7 setzen
599D   20 98 55   JSR $5598    CR out
59A0   A6 16      LDX $16      1 Zeile listen
59A2   A5 17      LDA $17
59A4   20 23 51   JSR $5123
59A7   46 55      LSR $55      HELP-Flag löschen
59A9   4C 98 55   JMP $5598    CR out

****************************** LIST-Unterroutine für HELP

59AC   A6 62      LDX $62      High-Byte laden
59AE   98         TYA          Offset in (A)
59AF   18         CLC          LIST-Zeiger korrigieren
59B0   65 61      ADC $61
59B2   90 01      BCC $59B5
59B4   E8         INX
59B5   EC 0F 12   CPX $120F    > Fehler PC ?
59B8   D0 14      BNE $59CE    Nein, Skip
59BA   CD 0E 12   CMP $120E
59BD   90 0F      BCC $59CE    Nein, Skip
59BF   F0 0D      BEQ $59CE    Nein, Skip
59C1   46 55      LSR $55      HELP-Flag löschen
59C3   A9 12      LDA #$12     Code für REVERS
59C5   24 D7      BIT $D7      80-Zeichen-Modus ?
59C7   10 02      BPL $59CB    Nein, Skip
59C9   A9 02      LDA #$02     Code für UNTERSTREICHEN EIN
59CB   4C 0C 56   JMP $560C    Zeichen out
59CE   60         RTS

****************************** BASIC-Befehl GOSUB

59CF   20 1D 5A   JSR $5A1D    GOSUB-Werte auf Stack
59D2   20 86 03   JSR $0386    CHRGOT
59D5   20 DB 59   JSR $59DB    GOTO
```

```
59D8   4C F6 4A   JMP $4AF6     Interpreterschleife

****************************** BASIC-Befehl GOTO

59DB   20 A0 50   JSR $50A0     Zeilennummer lesen
59DE   A5 0A      LDA $0A       Ziffern gelesen ?
59E0   F0 38      BEQ $5A1A     Nein, Fehler
59E2   20 A5 52   JSR $52A5     Zeilenende suchen
59E5   38         SEC
59E6   A5 3B      LDA $3B       Zeilennummer kleiner als
59E8   E5 16      SBC $16       laufende Zeile ?
59EA   A5 3C      LDA $3C
59EC   E5 17      SBC $17
59EE   B0 0B      BCS $59FB     nein, Skip
59F0   98         TYA           ab laufender Zeile suchen
59F1   38         SEC
59F2   65 3D      ADC $3D
59F4   A6 3E      LDX $3E
59F6   90 07      BCC $59FF
59F8   E8         INX
59F9   B0 04      BCS $59FF
59FB   A5 2D      LDA $2D       ab Programmstart suchen
59FD   A6 2E      LDX $2E
59FF   20 68 50   JSR $5068     Zeile suchen
5A02   90 11      BCC $5A15     Wenn nicht gefunden, Fehler
5A04   A5 61      LDA $61       PC setzen
5A06   E9 01      SBC #$01
5A08   85 3D      STA $3D
5A0A   A5 62      LDA $62
5A0C   E9 00      SBC #$00
5A0E   85 3E      STA $3E
5A10   24 7F      BIT $7F       Direktmodus ?
5A12   10 6D      BPL $5A81     Ja, Skip
5A14   60         RTS
5A15   A2 11      LDX #$11      'UNDEF'D STATEMENT'
5A17   4C 3C 4D   JMP $4D3C
5A1A   4C 6C 79   JMP $796C     'SYNTAX ERROR'

****************************** GOSUB-Werte auf BASIC-Stack legen

5A1D   A9 05      LDA #$05      BASIC-Stackpointer um 5 Bytes versetzen
5A1F   20 FE 4F   JSR $4FFE
5A22   A0 04      LDY #$04
5A24   A5 3E      LDA $3E       PC eintragen
5A26   91 7D      STA ($7D),Y
5A28   88         DEY
5A29   A5 3D      LDA $3D
5A2B   91 7D      STA ($7D),Y
```

```
5A2D   88         DEY
5A2E   A5 3C      LDA $3C       Zeilennummer eintragen
5A30   91 7D      STA ($7D),Y
5A32   88         DEY
5A33   A5 3B      LDA $3B
5A35   91 7D      STA ($7D),Y
5A37   88         DEY
5A38   A9 8D      LDA #$8D      Token für GOSUB eintragen
5A3A   91 7D      STA ($7D),Y
5A3C   60         RTS

****************************** Test auf GO TO und GO64

5A3D   20 80 03   JSR $0380     CHRGET
5A40   C9 A4      CMP #$A4      Token für TO ?
5A42   D0 06      BNE $5A4A     Nein, Skip
5A44   20 80 03   JSR $0380     CHRGET
5A47   4C DB 59   JMP $59DB     GOTO-Befehl
5A4A   20 F4 87   JSR $87F4     Byte-Wert holen
5A4D   E0 40      CPX #$40      = 64 ?
5A4F   F0 03      BEQ $5A54     Ja, Skip
5A51   4C 6C 79   JMP $796C     'SYNTAX ERROR'
5A54   20 E1 A7   JSR $A7E1     'ARE YOU SURE?'
5A57   D0 06      BNE $5A5F     Nein, nicht sicher
5A59   20 45 A8   JSR $A845     ROMs einschalten
5A5C   4C 4D FF   JMP $FF4D     C64-Modus aufrufen
5A5F   60         RTS


****************************** BASIC-Befehl CONT

5A60   D0 38      BNE $5A9A     Kein Trennzeichen, Skip
5A62   24 7F      BIT $7F       Direktmodus ?
5A64   30 34      BMI $5A9A     Ja, Skip
5A66   A2 1A      LDX #$1A      Nummer für 'CAN'T CONTINUE'
5A68   AC 03 12   LDY $1203     Contnummer gültig ?
5A6B   D0 03      BNE $5A70     Ja, Skip
5A6D   4C 3C 4D   JMP $4D3C     Fehlerausgabe
5A70   AD 02 12   LDA $1202     PC setzen
5A73   85 3D      STA $3D
5A75   84 3E      STY $3E
5A77   AD 00 12   LDA $1200     Zeilennummer setzen
5A7A   AC 01 12   LDY $1201
5A7D   85 3B      STA $3B
5A7F   84 3C      STY $3C

5A81   A9 80      LDA #$80      Programmmodus setzen
5A83   85 7F      STA $7F
```

```
5A85   0A         ASL
5A86   85 74      STA $74      AUTO-Offset auf 0
5A88   85 75      STA $75
5A8A   8D 7F 12   STA $127F    Interruptmöglichkeiten löschen
5A8D   85 F6      STA $F6      Auto-Insert löschen
5A8F   A2 02      LDX #$02     Interrupt-Speicher löschen
5A91   9D 76 12   STA $1276,X
5A94   CA         DEX
5A95   10 FA      BPL $5A91
5A97   4C 90 FF   JMP $FF90    Systemmeldungen verbieten
5A9A   60         RTS

******************************** BASIC-Befehl RUN

5A9B   F0 18      BEQ $5AB5    RUN ohne Angaben
5A9D   90 1C      BCC $5ABB    RUN mit Nummer
5A9F   A9 40      LDA #$40     Direktmodusflag vorbesetzen
5AA1   85 7F      STA $7F
5AA3   20 A7 A1   JSR $A1A7    LOAD Programm
5AA6   20 81 5A   JSR $5A81    Programm-Modus setzen
5AA9   20 F3 51   JSR $51F3    Zeiger auf Programmstart, CLR
5AAC   20 4F 4F   JSR $4F4F    Verkettung korrigieren
5AAF   20 98 55   JSR $5598    CR out
5AB2   4C F6 4A   JMP $4AF6    Interpreterschleife

5AB5   20 81 5A   JSR $5A81    Programm-Modus setzen
5AB8   4C F3 51   JMP $51F3    Zeiger auf Programmstart, CLR

5ABB   20 FA 51   JSR $51FA    CLR-Befehl
5ABE   20 86 03   JSR $0386    CHRGOT
5AC1   20 DB 59   JSR $59DB    GOTO-Befehl
5AC4   20 81 5A   JSR $5A81    Programm-Modus setzen
5AC7   4C F6 4A   JMP $4AF6    Interpreterschleife

******************************** BASIC-Befehl RESTORE

5ACA   F0 15      BEQ $5AE1    Wenn keine Nummer, Skip
5ACC   20 12 88   JSR $8812    Adressausdruck einlesen
5ACF   84 16      STY $16      Zeilennummer setzen
5AD1   85 17      STA $17
5AD3   20 64 50   JSR $5064    Zeile suchen
5AD6   B0 03      BCS $5ADB    Wenn gefunden, Skip
5AD8   4C 15 5A   JMP $5A15    'UNDEF'D STATEMENT'
5ADB   A5 61      LDA $61      Zeilenadresse nehmen
5ADD   A4 62      LDY $62
5ADF   B0 05      BCS $5AE6
5AE1   38         SEC          Programmstart nehmen
5AE2   A5 2D      LDA $2D
```

```
5AE4   A4 2E      LDY $2E
5AE6   E9 01      SBC #$01
5AE8   B0 01      BCS $5AEB
5AEA   88         DEY
5AEB   85 43      STA $43       in DATA-Adresse setzen
5AED   84 44      STY $44
5AEF   60         RTS

******************************* Token für RENUMBER

5AF0   89 8A 8D A7 8C D6 D7 D5 GOTO,RUN,GOSUB,THEN,RESTORE
                               RESUME,TRAP,ELSE

******************************* BASIC-Befehl RENUMBER

5AF8   20 F0 84   JSR $84F0     Test auf Programm-Modus
5AFB   A9 00      LDA #$00
5AFD   A2 0A      LDX #$0A
5AFF   8E 70 11   STX $1170     Startzeile auf 10
5B02   8D 71 11   STA $1171
5B05   8E 72 11   STX $1172     Schrittweite auf 10
5B08   8D 73 11   STA $1173
5B0B   85 5C      STA $5C       Anfangszeilennummer auf 0
5B0D   85 5D      STA $5D
5B0F   20 86 03   JSR $0386     CHRGOT
5B12   F0 54      BEQ $5B68
5B14   20 A0 50   JSR $50A0     Zeilennummer holen
5B17   A5 0A      LDA $0A       Zeilennummer vorhanden ?
5B19   F0 0A      BEQ $5B25     Nein, Skip ?
5B1B   A5 16      LDA $16       Startzeile setzen
5B1D   A6 17      LDX $17
5B1F   8D 70 11   STA $1170
5B22   8E 71 11   STX $1171
5B25   20 06 9E   JSR $9E06     Schrittweite auswerten und in (A)/(Y)
5B28   90 0E      BCC $5B38     Wenn keine Angabe, Skip
5B2A   8C 72 11   STY $1172     Schrittweite setzen
5B2D   8D 73 11   STA $1173
5B30   0D 72 11   ORA $1172     Schrittweite = 0 ?
5B33   D0 03      BNE $5B38     Nein, Skip
5B35   4C 28 7D   JMP $7D28     'ILLEGAL QUANTITY'
5B38    20 06 9E    JSR $9E06     Anfangszeilennummer auswerten und in
(A)/(Y)
5B3B   90 2B      BCC $5B68     Wenn keine Angabe, Skip
5B3D   84 5C      STY $5C       Zeilennummer setzen
5B3F   84 16      STY $16
5B41   85 5D      STA $5D
5B43   85 17      STA $17
5B45   20 64 50   JSR $5064     Programmzeile suchen
```

```
5B48   A5 61      LDA $61      Adresse setzen
5B4A   A6 62      LDX $62
5B4C   85 5A      STA $5A
5B4E   86 5B      STX $5B
5B50   AD 70 11   LDA $1170    Startzeilennummer setzen
5B53   AE 71 11   LDX $1171
5B56   85 16      STA $16
5B58   86 17      STX $17
5B5A   20 64 50   JSR $5064    Programmzeile suchen
5B5D   38         SEC
5B5E   A5 61      LDA $61      Startadresse Startzeile
5B60   E5 5A      SBC $5A      kleiner als
5B62   A5 62      LDA $62      Startadresse Anfangszeile
5B64   E5 5B      SBC $5B
5B66   90 CD      BCC $5B35    Ja, Fehler
5B68   20 68 5D   JSR $5D68    Zeiger auf Start setzen
5B6B   20 9C 5D   JSR $5D9C    Ein Zeichen überlesen
5B6E   C8         INY
5B6F   20 C9 03   JSR $03C9    Linkadresse-High = 0 ?
5B72   F0 3A      BEQ $5BAE    Ja, Skip
5B74   C8         INY
5B75   20 C9 03   JSR $03C9    Anfangszeile erreicht ?
5B78   38         SEC
5B79   E5 5C      SBC $5C
5B7B   C8         INY
5B7C   20 C9 03   JSR $03C9
5B7F   E5 5D      SBC $5D
5B81   B0 07      BCS $5B8A    Ja, Skip
5B83   20 9D 5B   JSR $5B9D    PC auf Linkadresse setzen
5B86   D0 EC      BNE $5B74
5B88   F0 24      BEQ $5BAE    Wenn Programmende, Skip

5B8A   20 9D 5B   JSR $5B9D    PC auf Linkadresse setzen
5B8D   F0 1F      BEQ $5BAE    Wenn Programmende, Skip
5B8F   20 89 5D   JSR $5D89    Zeilennummer erhöhen
5B92   B0 04      BCS $5B98    Wenn zu groß, Fehler
5B94   C9 F9      CMP #$F9
5B96   90 F2      BCC $5B8A    sonst Bereich weitertesten
5B98   A2 26      LDX #$26     'LINE NUMBER TOO LARGE'
5B9A   4C 3C 4D   JMP $4D3C

******************************** PC auf Linkadresse setzen

5B9D   A0 00      LDY #$00
5B9F   20 C9 03   JSR $03C9
5BA2   AA         TAX
5BA3   C8         INY
5BA4   20 C9 03   JSR $03C9    Programmende ?
```

```
5BA7   F0 04      BEQ $5BAD     Ja, Skip
5BA9   86 3D      STX $3D       PC neu setzen
5BAB   85 3E      STA $3E
5BAD   60         RTS

******************************

5BAE   A9 01      LDA #$01      Flag auf Testen setzen
5BB0   85 77      STA $77
5BB2   AD 10 12   LDA $1210     Programmendezeiger setzen
5BB5   AE 11 12   LDX $1211
5BB8   85 3F      STA $3F
5BBA   86 40      STX $40
5BBC   20 FB 5B   JSR $5BFB     RENUMBER auf Plausibilität prüfen
5BBF   C6 77      DEC $77       Flag auf RENUMBER setzen
5BC1   20 FB 5B   JSR $5BFB     Programmtext renumerieren

5BC4   20 99 5D   JSR $5D99     Linkadresse lesen
5BC7   F0 2F      BEQ $5BF8     Wenn Programmende, Skip
5BC9   20 9C 5D   JSR $5D9C     Zeilennummer-Low holen
5BCC   85 16      STA $16       und setzen
5BCE   C8         INY
5BCF   20 C9 03   JSR $03C9     Zeilennummer-High holen
5BD2   38         SEC           Aktuelle Zeile kleiner als
5BD3   E5 5D      SBC $5D       Startzeile ?
5BD5   90 19      BCC $5BF0     Ja, Skip
5BD7   D0 06      BNE $5BDF     >, Skip
5BD9   A5 16      LDA $16       Aktuelle Zeile kleiner als
5BDB   E5 5C      SBC $5C       Startzeile ?
5BDD   90 11      BCC $5BF0     Ja, Skip
5BDF   A5 64      LDA $64       Zeile mit neuer Nummer
5BE1   91 3D      STA ($3D),Y   versehen
5BE3   88         DEY
5BE4   A5 65      LDA $65
5BE6   91 3D      STA ($3D),Y
5BE8   20 9C 5D   JSR $5D9C     Zeile überlesen
5BEB   20 80 5D   JSR $5D80
5BEE   F0 D4      BEQ $5BC4     Unbedingter Sprung

5BF0   20 9C 5D   JSR $5D9C     Zeile überlesen
5BF3   20 83 5D   JSR $5D83
5BF6   F0 CC      BEQ $5BC4     Unbedingter Sprung

5BF8   4C E5 5E   JMP $5EE5     Programmende neu setzen und 'READY.'

******************************

5BFB   20 54 52   JSR $5254     PC auf Programmstart
```

```
5BFE   20 99 5D   JSR $5D99     2 Zeichen lesen
5C01   D0 03      BNE $5C06     Wenn Linkadresse High gesetzt, Skip
5C03   4C 68 5D   JMP $5D68     Zeiger auf Start, Ende
5C06   20 9C 5D   JSR $5D9C     Zeilennummer setzen
5C09   85 4B      STA $4B
5C0B   20 9C 5D   JSR $5D9C
5C0E   85 4C      STA $4C
5C10   20 9C 5D   JSR $5D9C     Zeichen aus Programmtext holen
5C13   C9 22      CMP #$22      '"' ?
5C15   D0 0B      BNE $5C22     Nein, Skip
5C17   20 9C 5D   JSR $5D9C     Zeichen aus Programmtext holen
5C1A   F0 E2      BEQ $5BFE     Wenn Zeilenende, nächste Zeile testen
5C1C   C9 22      CMP #$22      2. '"' erreicht ?
5C1E   D0 F7      BNE $5C17     Nein, Zeichen überlesen
5C20   F0 EE      BEQ $5C10     Ja, Zeile weitertesten

5C22   AA         TAX           Zeilenende ?
5C23   F0 D9      BEQ $5BFE     Ja, nächste Zeile testen
5C25   10 E9      BPL $5C10     Wenn kein Token, überlesen
5C27   A2 08      LDX #$08      Token in RENUMBER-Tabelle ?
5C29   DD EF 5A   CMP $5AEF,X
5C2C   F0 28      BEQ $5C56     Ja, Skip
5C2E   CA         DEX
5C2F   D0 F8      BNE $5C29
5C31   C9 CB      CMP #$CB      Token für GO ?
5C33   D0 0B      BNE $5C40     Nein, Skip
5C35   20 80 03   JSR $0380     Nächstes Zeichen Zeilenende ?
5C38   F0 C4      BEQ $5BFE     Ja, nächste Zeile testen
                                (Fehler, reagiert auch bei ':' !)
5C3A   C9 A4      CMP #$A4      Token für TO ?
5C3C   F0 18      BEQ $5C56     Ja, Skip
5C3E   D0 D0      BNE $5C10     Nein, überlesen

5C40   C9 FE      CMP #$FE      Sondertoken 1 ?
5C42   D0 CC      BNE $5C10     Nein, überlesen
5C44   20 80 03   JSR $0380     Nächstes Zeichen Trennzeichen ?
5C47   F0 EC      BEQ $5C35     Ja, überlesen (Sinn sehr fragwürdig !)
5C49   C9 17      CMP #$17      Token für COLLISION ?
5C4B   D0 C3      BNE $5C10
5C4D   20 80 03   JSR $0380     Zeichen bis ',' überlesen
5C50   F0 E3      BEQ $5C35     Wenn Trennzeichen, weitertesten
5C52   C9 2C      CMP #$2C      ',' ?
5C54   D0 F7      BNE $5C4D     Nein, überlesen

5C56   A5 3D      LDA $3D       PC retten
5C58   8D 00 12   STA $1200
5C5B   A5 3E      LDA $3E
5C5D   8D 01 12   STA $1201
```

```
5C60   20 80 03   JSR $0380     CHRGET
5C63   B0 AE      BCS $5C13     Wenn keine Zahl, überlesen
5C65   20 A0 50   JSR $50A0     Zeilennummer einlesen
5C68   20 19 5D   JSR $5D19     Neue Zeilennummer berechnen
5C6B   AD 00 12   LDA $1200     PC wieder holen
5C6E   85 3D      STA $3D
5C70   AD 01 12   LDA $1201
5C73   85 3E      STA $3E
5C75   20 80 03   JSR $0380     CHRGET Spaces überlesen
5C78   A5 3D      LDA $3D       PC um 1 erniedrigen
5C7A   D0 02      BNE $5C7E
5C7C   C6 3E      DEC $3E
5C7E   C6 3D      DEC $3D
5C80   A2 FF      LDX #$FF      Pufferoffset auf Start
5C82   A5 77      LDA $77       Nummer jetzt eintragen ?
5C84   F0 41      BEQ $5CC7     Ja, Skip
5C86   20 8F 5C   JSR $5C8F     Speichergröße testen
5C89   C9 2C      CMP #$2C      Nächstes Zeichen ',' ?
5C8B   F0 C9      BEQ $5C56     Ja, weiter Zeilennummer verarbeiten
5C8D   D0 84      BNE $5C13     Nein, weitertesten

******************************* Speicher für Zeilennummern testen

5C8F   E8         INX
5C90   BD 01 01   LDA $0101,X   Pufferende erreicht ?
5C93   F0 1F      BEQ $5CB4     Ja, Skip
5C95   20 80 03   JSR $0380     Zeichen aus Programmtext eine Zahl ?
5C98   90 F5      BCC $5C8F     Ja, weitertesten
5C9A   E6 3F      INC $3F       Es muß ein Zeichen mehr in den
5C9C   D0 02      BNE $5CA0     Programmtext übertragen werden
5C9E   E6 40      INC $40
5CA0   38         SEC
5CA1   A5 3F      LDA $3F       Speicherende erreicht ?
5CA3   ED 12 12   SBC $1212
5CA6   A5 40      LDA $40
5CA8   ED 13 12   SBC $1213
5CAB   B0 17      BCS $5CC4     Ja, Fehler
5CAD   E8         INX
5CAE   BD 01 01   LDA $0101,X   Pufferende jetzt erreicht ?
5CB1   D0 E7      BNE $5C9A     Nein, noch ein Zeichen mehr
5CB3   60         RTS

5CB4   20 80 03   JSR $0380     Nächstes Zeichen aus Programm eine Zahl ?
5CB7   B0 0A      BCS $5CC3     Nein, Ende
5CB9   A5 3F      LDA $3F       Es kann ein Zeichen aus dem Programm
5CBB   D0 02      BNE $5CBF     gelöscht werden
5CBD   C6 40      DEC $40
5CBF   C6 3F      DEC $3F
```

```
5CC1   90 F1      BCC $5CB4    Unbedingter Sprung, weitertesten
5CC3   60         RTS
5CC4   4C 3A 4D   JMP $4D3A    'OUT OF MEMORY'

****************************** Neue Zeilennummer eintragen

5CC7   E8         INX          Offset erhöhen
5CC8   BD 01 01   LDA $0101,X  Pufferende erreicht ?
5CCB   F0 2C      BEQ $5CF9    Ja, Skip
5CCD   48         PHA          Zeichen retten
5CCE   20 9C 5D   JSR $5D9C    Zeichen aus Programmtext holen
5CD1   C9 3A      CMP #$3A     Zeichen > '9' ?
5CD3   B0 0C      BCS $5CE1    Ja, Skip
5CD5   C9 20      CMP #$20     Space ?
5CD7   F0 08      BEQ $5CE1    Ja, Skip
5CD9   38         SEC
5CDA   E9 30      SBC #$30
5CDC   38         SEC
5CDD   E9 D0      SBC #$D0     Zeichen < '0' ?
5CDF   90 10      BCC $5CF1    Nein, Ziffer, überschreiben
5CE1   20 A7 5D   JSR $5DA7    Zeiger vorbesetzen
5CE4   E6 6D      INC $6D      Ein Zeichen im Programm einfügen
5CE6   20 DF 5D   JSR $5DDF
5CE9   EE 10 12   INC $1210    Programmende korrigieren
5CEC   D0 03      BNE $5CF1
5CEE   EE 11 12   INC $1211
5CF1   68         PLA          Ziffer aus Puffer holen
5CF2   A0 00      LDY #$00
5CF4   91 3D      STA ($3D),Y  und in Programmtext eintragen
5CF6   E8         INX          Puffer
5CF7   D0 CF      BNE $5CC8    weiterbearbeiten
5CF9   20 80 03   JSR $0380    Nächstes Zeichen noch eine Ziffer ?
5CFC   B0 8B      BCS $5C89    Nein, weitertesten
5CFE   20 A7 5D   JSR $5DA7    Zeiger vorbesetzen
5D01   C6 6D      DEC $6D      Programm um ein Zeichen kürzen
5D03   20 C6 5D   JSR $5DC6
5D06   AD 10 12   LDA $1210    Programmende korrigieren
5D09   D0 03      BNE $5D0E
5D0B   CE 11 12   DEC $1211
5D0E   CE 10 12   DEC $1210
5D11   20 86 03   JSR $0386    Noch eine Ziffer ?
5D14   90 E8      BCC $5CFE    Ja, löschen
5D16   4C 89 5C   JMP $5C89    Weitertesten

****************************** Unterroutine für RENUMBER

5D19   20 68 5D   JSR $5D68    Zeiger auf Start
5D1C   20 99 5D   JSR $5D99    Linkadresse lesen
```

```
5D1F   D0 0D      BNE $5D2E    Wenn kein Programmende, Skip
5D21   A2 27      LDX #$27     'UNRESOLVED REFERENCE'
5D23   A5 4B      LDA $4B      Zeilennummer setzen
5D25   85 3B      STA $3B
5D27   A5 4C      LDA $4C
5D29   85 3C      STA $3C
5D2B   4C 3C 4D   JMP $4D3C    Fehlerausgabe
5D2E   20 9C 5D   JSR $5D9C    Zeilennummer Low holen
5D31   85 5A      STA $5A      und setzen
5D33   C5 16      CMP $16      = Suchzeile ?
5D35   D0 27      BNE $5D5E    Nein, Skip
5D37   20 9C 5D   JSR $5D9C    Zeilennummer High holen
5D3A   85 5B      STA $5B      und setzen
5D3C   C5 17      CMP $17      = Suchzeile ?
5D3E   D0 23      BNE $5D63    Nein, Skip
5D40   38         SEC          Suchzeile
5D41   E5 5D      SBC $5D      kleiner als Startzeile ?
5D43   90 08      BCC $5D4D    Ja, Suchzeile setzen
5D45   D0 0E      BNE $5D55    Falls > zur Umwandlung
5D47   A5 16      LDA $16
5D49   E5 5C      SBC $5C
5D4B   B0 08      BCS $5D55    Falls >= zur Umwandlung
5D4D   A5 16      LDA $16      Suchzeilennummer setzen
5D4F   85 65      STA $65
5D51   A5 17      LDA $17
5D53   85 64      STA $64
5D55   A2 90      LDX #$90
5D57   38         SEC
5D58   20 75 8C   JSR $8C75    Adresswert in FAC#1
5D5B   4C 42 8E   JMP $8E42    FAC#1 in ASCII ab $0100

5D5E   20 9C 5D   JSR $5D9C    Zeilennummer High lesen
5D61   85 5B      STA $5B      und setzen
5D63   20 75 5D   JSR $5D75    Falls nötig Zeilennummer erhöhen
5D66   F0 B4      BEQ $5D1C    Unbedingter Sprung

****************************** Zeiger auf Startposition

5D68   AD 70 11   LDA $1170    Startzeilennummer setzen
5D6B   85 65      STA $65
5D6D   AD 71 11   LDA $1171
5D70   85 64      STA $64
5D72   4C 54 52   JMP $5254    PC auf Programmstart

****************************** Zeilenleseroutinen für RENUMBER

5D75   A5 5A      LDA $5A      Aktuelle Zeile
5D77   38         SEC          kleiner als
```

```
5D78   E5 5C      SBC $5C      Startzeile ?
5D7A   A5 5B      LDA $5B
5D7C   E5 5D      SBC $5D
5D7E   90 03      BCC $5D83    Ja, Skip
5D80   20 89 5D   JSR $5D89    Zeilennummer erhöhen
5D83   20 9C 5D   JSR $5D9C    Zeilenende erreicht ?
5D86   D0 FB      BNE $5D83    Nein, weitersuchen
5D88   60         RTS

****************************** Zeilennummer um Schrittweite erhöhen

5D89   A5 65      LDA $65      Aktuelle neue Nummer
5D8B   18         CLC
5D8C   6D 72 11   ADC $1172    + Schrittweite
5D8F   85 65      STA $65
5D91   A5 64      LDA $64
5D93   6D 73 11   ADC $1173
5D96   85 64      STA $64
5D98   60         RTS

****************************** 2.Zeichen aus Programmtext holen

5D99   20 9C 5D   JSR $5D9C    1 Zeichen überlesen
5D9C   A0 00      LDY #$00     GETCHR aus Programmtext ohne
5D9E   E6 3D      INC $3D      Untersuchung des Zeichens
5DA0   D0 02      BNE $5DA4
5DA2   E6 3E      INC $3E
5DA4   4C C9 03   JMP $03C9

****************************** Zeiger vorbesetzen für Move

5DA7   A5 3D      LDA $3D      ($24) auf PC
5DA9   85 24      STA $24
5DAB   A5 3E      LDA $3E
5DAD   85 25      STA $25
5DAF   AD 10 12   LDA $1210    ($26) auf Programmende
5DB2   85 26      STA $26
5DB4   AD 11 12   LDA $1211
5DB7   85 27      STA $27
5DB9   A0 00      LDY #$00     Offsets löschen
5DBB   84 0D      STY $0D
5DBD   84 6D      STY $6D
5DBF   60         RTS

****************************** MOVEDOWN Bank 0, Einsprung $5DC6

5DC0   E6 24      INC $24      Movepointer erhöhen
5DC2   D0 02      BNE $5DC6
```

```
5DC4   E6 25      INC $25

5DC6   A4 0D      LDY $0D       Offset 1 laden
5DC8   C8         INY
5DC9   20 05 43   JSR $4305     Zeichen aus Bank 0 holen
5DCC   A4 6D      LDY $6D       Offset 2 laden
5DCE   C8         INY
5DCF   91 24      STA ($24),Y   Zeichen in Bank 1 schreiben
5DD1   20 EE 5D   JSR $5DEE     Ende ?
5DD4   D0 EA      BNE $5DC0     Nein, weitermachen
5DD6   60         RTS

****************************** MOVEUP Bank 1, Einsprung $5DDF

5DD7   A5 26      LDA $26       Movepointer erniedrigen
5DD9   D0 02      BNE $5DDD
5DDB   C6 27      DEC $27
5DDD   C6 26      DEC $26

5DDF   A4 0D      LDY $0D       Offset 1 laden
5DE1   20 C0 03   JSR $03C0     Zeichen aus Bank 0 holen
5DE4   A4 6D      LDY $6D       Offset 2 laden
5DE6   91 26      STA ($26),Y   Zeichen in Bank 0 speichern
5DE8   20 EE 5D   JSR $5DEE     Ende ?
5DEB   D0 EA      BNE $5DD7     Nein, weitermachen
5DED   60         RTS

****************************** Test auf ($24)=($26)

5DEE   A5 24      LDA $24
5DF0   C5 26      CMP $26
5DF2   D0 04      BNE $5DF8
5DF4   A5 25      LDA $25
5DF6   C5 27      CMP $27
5DF8   60         RTS

****************************** BASIC-Befehl FOR

5DF9   A9 80      LDA #$80      Integerwerte sperren
5DFB   85 12      STA $12
5DFD   20 C6 53   JSR $53C6     FOR-Variable durch LET vorbesetzen
5E00   A9 81      LDA #$81      Token für FOR
5E02   20 AA 4F   JSR $4FAA     in BASIC-Stack ?
5E05   F0 08      BEQ $5E0F     Ja, Skip
5E07   A9 12      LDA #$12      Platz im Stack reservieren
5E09   20 FE 4F   JSR $4FFE
5E0C   20 47 50   JSR $5047     BASIC-Stackpointer in Hilfszeiger
5E0F   20 50 50   JSR $5050     Hilfszeiger in BASIC-Stackpointer
```

```
5E12   20 A2 52   JSR $52A2     Nächstes Trennzeichen suchen
5E15   98         TYA
5E16   A0 11      LDY #$11
5E18   18         CLC
5E19   65 3D      ADC $3D       PC in BASIC-Stack
5E1B   91 7D      STA ($7D),Y
5E1D   A5 3E      LDA $3E
5E1F   69 00      ADC #$00
5E21   88         DEY
5E22   91 7D      STA ($7D),Y
5E24   A5 3C      LDA $3C       Zeilennummer in BASIC-Stack
5E26   88         DEY
5E27   91 7D      STA ($7D),Y
5E29   A5 3B      LDA $3B
5E2B   88         DEY
5E2C   91 7D      STA ($7D),Y
5E2E   A9 A4      LDA #$A4      Token für TO
5E30   20 5E 79   JSR $795E     Prüft auf Code
5E33   20 DA 77   JSR $77DA     Prüft auf numerische Variable
5E36   20 D7 77   JSR $77D7     Ausdruck in FAC#1
5E39   A5 68      LDA $68
5E3B   09 7F      ORA #$7F
5E3D   25 64      AND $64
5E3F   85 64      STA $64
5E41   A2 04      LDX #$04      Schleifenendwert auf BASIC-Stack
5E43   A0 0D      LDY #$0D
5E45   B5 63      LDA $63,X
5E47   91 7D      STA ($7D),Y
5E49   CA         DEX
5E4A   88         DEY
5E4B   10 F8      BPL $5E45
5E4D   A9 9C      LDA #$9C      Zeiger auf Konstante 1
5E4F   A0 89      LDY #$89
5E51   20 D4 8B   JSR $8BD4     als STEP-Wert in FAC#1
5E54   20 86 03   JSR $0386     CHRGOT
5E57   C9 A9      CMP #$A9      = Token für STEP ?
5E59   D0 06      BNE $5E61     Nein, Skip
5E5B   20 80 03   JSR $0380     CHRGET
5E5E   20 D7 77   JSR $77D7     Ausdruck in FAC#1
5E61   20 57 8C   JSR $8C57     Vorzeichen von FAC#1 holen
5E64   48         PHA           und retten
5E65   20 47 8C   JSR $8C47     FAC#1 runden
5E68   68         PLA           Vorzeichen wieder holen
5E69   A0 08      LDY #$08      STEP-Wert auf BASIC-Stack
5E6B   A2 05      LDX #$05
5E6D   91 7D      STA ($7D),Y
5E6F   B5 62      LDA $62,X
5E71   88         DEY
```

```
5E72   CA         DEX
5E73   10 F8      BPL $5E6D
5E75   A5 4C      LDA $4C        Variablenadresse auf BASIC-Stack
5E77   91 7D      STA ($7D),Y
5E79   A5 4B      LDA $4B
5E7B   88         DEY
5E7C   91 7D      STA ($7D),Y
5E7E   A9 81      LDA #$81       FOR-Token auf BASIC-Stack
5E80   88         DEY
5E81   91 7D      STA ($7D),Y
5E83   60         RTS

5E84   4C 6C 79   JMP $796C      'SYNTAX'

****************************** BASIC-Befehl DELETE

5E87   20 F0 84   JSR $84F0      Test auf Programm-Modus
5E8A   20 86 03   JSR $0386      Letztes Zeichen = Trennzeichen ?
5E8D   F0 F5      BEQ $5E84      Ja, Skip
5E8F   20 FB 5E   JSR $5EFB      Zeilenbereich lesen
5E92   A5 61      LDA $61        Startzeilenadresse vorbesetzen
5E94   A6 62      LDX $62
5E96   85 26      STA $26
5E98   86 27      STX $27
5E9A   20 64 50   JSR $5064      Zeile suchen
5E9D   90 15      BCC $5EB4      Wenn nicht gefunden, Skip
5E9F   A0 01      LDY #$01       Linkadresse High laden
5EA1   20 EC 42   JSR $42EC
5EA4   88         DEY
5EA5   AA         TAX            Programmende ?
5EA6   D0 05      BNE $5EAD      Nein, Skip
5EA8   20 EC 42   JSR $42EC      Linkadresse Low laden
5EAB   F0 07      BEQ $5EB4      Wenn Programmende, Skip
5EAD   20 EC 42   JSR $42EC      Linkadresse Low laden
5EB0   85 61      STA $61        Adresse neu setzen
5EB2   86 62      STX $62
5EB4   A5 26      LDA $26        Löschung erforderlich ?
5EB6   38         SEC
5EB7   E5 61      SBC $61
5EB9   AA         TAX
5EBA   A5 27      LDA $27
5EBC   E5 62      SBC $62
5EBE   A8         TAY
5EBF   B0 24      BCS $5EE5      Nein, Skip
5EC1   8A         TXA
5EC2   18         CLC            Programmende korrigieren
5EC3   6D 10 12   ADC $1210
5EC6   8D 10 12   STA $1210
```

```
5EC9   98         TYA
5ECA   6D 11 12   ADC $1211
5ECD   8D 11 12   STA $1211
5ED0   A0 00      LDY #$00      Programmtext blockweise verschieben
5ED2   20 EC 42   JSR $42EC
5ED5   91 26      STA ($26),Y
5ED7   C8         INY
5ED8   D0 F8      BNE $5ED2
5EDA   E6 62      INC $62
5EDC   E6 27      INC $27
5EDE   AD 11 12   LDA $1211
5EE1   C5 27      CMP $27
5EE3   B0 ED      BCS $5ED2
5EE5   20 4F 4F   JSR $4F4F     Verkettung korrigieren
5EE8   A5 24      LDA $24       Programmende neu setzen
5EEA   A6 25      LDX $25
5EEC   18         CLC
5EED   69 02      ADC #$02
5EEF   8D 10 12   STA $1210
5EF2   90 01      BCC $5EF5
5EF4   E8         INX
5EF5   8E 11 12   STX $1211
5EF8   4C 37 4D   JMP $4D37     'READY.'

****************************** Zeilenbereich lesen

5EFB   F0 12      BEQ $5F0F     Wenn Trennzeichen, Skip
5EFD   90 10      BCC $5F0F     Wenn Zahl, Skip
5EFF   C9 AB      CMP #$AB      Token für '-' ?
5F01   D0 0C      BNE $5F0F     Nein, Skip
5F03   A0 01      LDY #$01
5F05   20 C9 03   JSR $03C9     Folgendes Zeichen = Trennzeichen ?
5F08   F0 27      BEQ $5F31     Ja, Fehler
5F0A   C9 3A      CMP #$3A      ':' ?
5F0C   F0 23      BEQ $5F31     Ja, Fehler
5F0E   38         SEC
5F0F   20 A0 50   JSR $50A0     Zeilennummer lesen
5F12   20 64 50   JSR $5064     Zeile suchen
5F15   20 86 03   JSR $0386     Momentanes Zeichen = Trennzeichen ?
5F18   F0 0C      BEQ $5F26     Ja, Skip
5F1A   C9 AB      CMP #$AB      = '-' ?
5F1C   D0 13      BNE $5F31     Nein, Fehler
5F1E   20 80 03   JSR $0380     CHRGET
5F21   20 A0 50   JSR $50A0     Zeilennummer holen
5F24   D0 0B      BNE $5F31     Wenn Fehler, Skip
5F26   A5 0A      LDA $0A       Zeilennummer gelesen
5F28   D0 06      BNE $5F30     Ja, Skip
5F2A   A9 FF      LDA #$FF      Zeilennummer auf Maximalwert
```

```
5F2C   85 16      STA $16
5F2E   85 17      STA $17
5F30   60         RTS
5F31   4C 6C 79   JMP $796C     'SYNTAX'

****************************** BASIC-Befehl PUDEF

5F34   20 7B 87   JSR $877B     FRMEVL + FRESTR
5F37   A8         TAY           Länge in (Y)
5F38   88         DEY
5F39   C0 04      CPY #$04      Länge > 4 ?
5F3B   90 03      BCC $5F40     Nein, Skip
5F3D   4C 28 7D   JMP $7D28     'ILLEGAL QUANTITY'
5F40   20 B7 03   JSR $03B7     Zeichen holen
5F43   8D 03 FF   STA $FF03     Write in Bank 0 setzen
5F46   99 04 12   STA $1204,Y   Stringwerte in PRINT USING Tabelle
5F49   88         DEY           übertragen
5F4A   10 F4      BPL $5F40
5F4C   60         RTS

****************************** BASIC-Befehl TRAP

5F4D   20 D9 84   JSR $84D9     Test auf Direktmodus
5F50   20 86 03   JSR $0386     CHRGOT
5F53   F0 07      BEQ $5F5C     Wenn Trennzeichen, Skip
5F55   20 12 88   JSR $8812     TRAP-Nummer holen
5F58   8C 0B 12   STY $120B     und setzen
5F5B   2C         .BYTE $2C
5F5C   A9 FF      LDA #$FF      TRAP-Nummer auf ungültig
5F5E   8D 0C 12   STA $120C
5F61   60         RTS

****************************** BASIC-Befehl RESUME

5F62   20 D9 84   JSR $84D9     Test auf Direktmodus
5F65   AE 0A 12   LDX $120A     Fehler aufgetreten ?
5F68   E8         INX
5F69   F0 70      BEQ $5FDB     Nein, 'CAN'T RESUME'
5F6B   20 86 03   JSR $0386     Nächstes Zeichen Trennzeichen ?
5F6E   F0 47      BEQ $5FB7     Ja, Skip
5F70   90 3A      BCC $5FAC     Wenn Zahl, Skip
5F72   C9 82      CMP #$82      Token für NEXT ?
5F74   D0 62      BNE $5FD8     Nein, Fehler
5F76   20 B7 5F   JSR $5FB7     TRAP-Daten in Workbereich übertragen
5F79   A0 00      LDY #$00
5F7B   20 C9 03   JSR $03C9     Zeilenende ?
5F7E   D0 26      BNE $5FA6     Nein, Skip
5F80   C8         INY
```

```
5F81    20 C9 03    JSR $03C9      Programmende ?
5F84    D0 09       BNE $5F8F      Nein, Skip
5F86    C8          INY
5F87    20 C9 03    JSR $03C9
5F8A    D0 03       BNE $5F8F      Nein, Skip
5F8C    4C 37 4D    JMP $4D37      'READY.'

5F8F    A0 03       LDY #$03
5F91    20 C9 03    JSR $03C9      Neue Zeilennummer setzen
5F94    85 3B       STA $3B
5F96    C8          INY
5F97    20 C9 03    JSR $03C9
5F9A    85 3C       STA $3C
5F9C    98          TYA
5F9D    18          CLC
5F9E    65 3D       ADC $3D        PC korrigieren
5FA0    85 3D       STA $3D
5FA2    90 02       BCC $5FA6
5FA4    E6 3E       INC $3E
5FA6    20 80 03    JSR $0380      CHRGET
5FA9    4C 8F 52    JMP $528F      DATA-Befehl

5FAC    20 12 88    JSR $8812      Adresse auswerten
5FAF    85 17       STA $17        setzen
5FB1    20 C6 5F    JSR $5FC6      TRAP-Daten löschen
5FB4    4C FB 59    JMP $59FB      und GOTO

****************************** TRAP-Daten in Workdaten übertragen
                               und TRAP-Daten löschen
5FB7    A2 01       LDX #$01
5FB9    BD 09 12    LDA $1209,X    Zeilennummer setzen
5FBC    95 3B       STA $3B,X
5FBE    BD 0E 12    LDA $120E,X    PC setzen
5FC1    95 3D       STA $3D,X
5FC3    CA          DEX
5FC4    10 F3       BPL $5FB9
5FC6    A2 FF       LDX #$FF       TRAP-Daten löschen
5FC8    8E 08 12    STX $1208
5FCB    8E 09 12    STX $1209
5FCE    8E 0A 12    STX $120A
5FD1    AE 0D 12    LDX $120D
5FD4    8E 0C 12    STX $120C
5FD7    60          RTS

5FD8    4C 6C 79    JMP $796C      'SYNTAX'

5FDB    A2 1F       LDX #$1F       'CAN'T RESUME'
```

```
5FDD   4C 3C 4D   JMP $4D3C

****************************** BASIC-Befehl DO

5FE0   A0 01      LDY #$01     Workdaten in Zwischenspeicher übertragen
5FE2   B9 3D 00   LDA $003D,Y
5FE5   99 14 12   STA $1214,Y
5FE8   B9 3B 00   LDA $003B,Y
5FEB   99 16 12   STA $1216,Y
5FEE   88         DEY
5FEF   10 F1      BPL $5FE2
5FF1   20 86 03   JSR $0386    Aktuelles Zeichen Trennzeichen ?
5FF4   F0 1C      BEQ $6012    Ja, Skip
5FF6   C9 FC      CMP #$FC     Token für UNTIL
5FF8   F0 11      BEQ $600B    Ja, Skip
5FFA   C9 FD      CMP #$FD     Token für WHILE
5FFC   D0 43      BNE $6041    Nein, Fehler
5FFE   20 DB 60   JSR $60DB    Logischen Ausdruck auswerten
6001   A5 63      LDA $63      Ausdruck falsch ?
6003   D0 0D      BNE $6012    Nein, Skip
6005   20 86 03   JSR $0386    CHRGOT
6008   4C 47 60   JMP $6047    Schleife überlesen
600B   20 DB 60   JSR $60DB    Logischen Ausdruck auswerten
600E   A5 63      LDA $63      Ausdruck wahr ?
6010   D0 F3      BNE $6005    Nein, Schleife überlesen

6012   A9 05      LDA #$05     5 Bytes in BASIC-Stack reservieren
6014   20 FE 4F   JSR $4FFE
6017   8D 03 FF   STA $FF03    Write in Bank 0 setzen
601A   A0 04      LDY #$04
601C   AD 15 12   LDA $1215    Zeilennummer auf Stapel
601F   91 7D      STA ($7D),Y
6021   88         DEY
6022   AD 14 12   LDA $1214
6025   91 7D      STA ($7D),Y
6027   88         DEY
6028   AD 17 12   LDA $1217    PC auf Stapel
602B   91 7D      STA ($7D),Y
602D   88         DEY
602E   AD 16 12   LDA $1216
6031   91 7D      STA ($7D),Y
6033   88         DEY
6034   A9 EB      LDA #$EB     Token für DO auf Stapel
6036   91 7D      STA ($7D),Y
6038   60         RTS
```

```
****************************** BASIC-Befehl EXIT

6039   20 9B 60   JSR $609B    LOOP-Werte vom Stapel holen
603C   20 86 03   JSR $0386    CHRGOT
603F   F0 06      BEQ $6047    Wenn Trennzeichen, Schleife überlesen
6041   4C 6C 79   JMP $796C    'SYNTAX'
6044   20 80 03   JSR $0380    CHRGET

6047   F0 17      BEQ $6060    Wenn Trennzeichen, Skip
6049   C9 EC      CMP #$EC     Token für LOOP ?
604B   F0 3A      BEQ $6087    Ja, Skip
604D   C9 22      CMP #$22     '"' ?
604F   F0 0A      BEQ $605B    Ja, Skip
6051   C9 EB      CMP #$EB     Token für DO ?
6053   D0 EF      BNE $6044    Nein, weiterlesen
6055   20 44 60   JSR $6044    Rekursiver Aufruf
6058   4C 05 60   JMP $6005    Weiter überlesen

605B   20 7C 53   JSR $537C    Text in '"' überlesen
605E   D0 E4      BNE $6044    Wenn kein Trennzeichen, weiter überlesen
6060   C9 3A      CMP #$3A     ':' ?
6062   F0 E0      BEQ $6044    Ja, überlesen
6064   24 7F      BIT $7F      Direktmodus ?
6066   10 42      BPL $60AA    Ja, Skip
6068   A0 02      LDY #$02
606A   20 C9 03   JSR $03C9    Programmende ?
606D   F0 3B      BEQ $60AA    Ja, Skip
606F   C8         INY
6070   20 C9 03   JSR $03C9    Neue Zeilennummer setzen
6073   85 3B      STA $3B
6075   C8         INY
6076   20 C9 03   JSR $03C9
6079   85 3C      STA $3C
607B   98         TYA
607C   18         CLC
607D   65 3D      ADC $3D      PC korrigieren
607F   85 3D      STA $3D
6081   90 C1      BCC $6044    Weiter überlesen
6083   E6 3E      INC $3E
6085   D0 BD      BNE $6044    Weiter überlesen
6087   4C 8F 52   JMP $528F    DATA-Befehl

****************************** BASIC-Befehl LOOP

608A   F0 35      BEQ $60C1    Wenn Trennzeichen, Skip
608C   C9 FD      CMP #$FD     Token für WHILE ?
608E   F0 2C      BEQ $60BC    Ja, Skip
6090   C9 FC      CMP #$FC     Token für UNTIL
```

```
6092   D0 AD      BNE $6041     Nein, Fehler
6094   20 DB 60   JSR $60DB     Logischen Ausdruck auswerten
6097   A5 63      LDA $63       Ausdruck gleich 0 (false) ?
6099   F0 26      BEQ $60C1     Ja, Skip

****************************** LOOP-Daten vom BASIC-Stack löschen

609B   A9 EB      LDA #$EB      Token für DO
609D   20 AA 4F   JSR $4FAA     im BASIC-Stack suchen
60A0   D0 15      BNE $60B7     Wenn nicht gefunden, Fehler
60A2   20 50 50   JSR $5050     Stackpointer setzen
60A5   A0 05      LDY #$05      BASIC-Stack um 5 Bytes leeren
60A7   4C 59 50   JMP $5059

60AA   AD 16 12   LDA $1216     Aktuelle Werte in Zeilennummer
60AD   AE 17 12   LDX $1217
60B0   85 3B      STA $3B
60B2   86 3C      STX $3C
60B4   A2 20      LDX #$20      'LOOP NOT FOUND'
60B6   2C         .BYTE $2C
60B7   A2 21      LDX #$21      'LOOP WITHOUT DO'
60B9   4C 3C 4D   JMP $4D3C

60BC   20 DB 60   JSR $60DB     Logischen Ausdruck auswerten
60BF   F0 DA      BEQ $609B     Wenn Ausdruck falsch, LOOP beenden
60C1   20 9B 60   JSR $609B     LOOP-Daten vom Stapel holen
60C4   88         DEY
60C5   B1 3F      LDA ($3F),Y   PC vom Stapel holen
60C7   85 3E      STA $3E
60C9   88         DEY
60CA   B1 3F      LDA ($3F),Y
60CC   85 3D      STA $3D
60CE   88         DEY
60CF   B1 3F      LDA ($3F),Y   Zeilennummer vom Stapel holen
60D1   20 3B A8   JSR $A83B
60D4   B1 3F      LDA ($3F),Y
60D6   85 3B      STA $3B
60D8   4C E0 5F   JMP $5FE0     DO aufrufen

****************************** Logischen Ausdruck auswerten

60DB   20 80 03   JSR $0380     CHRGET
60DE   4C EF 77   JMP $77EF     FRMEVL Ausdruck auswerten

****************************** BASIC-Routine KEY-Definition

60E1   20 F4 87   JSR $87F4     Byte-Wert holen
60E4   CA         DEX
```

```
60E5    E0 08       CPX #$08      > 8 ?
60E7    90 03       BCC $60EC     Nein, Skip
60E9    4C 28 7D    JMP $7D28     'ILLEGAL QUANTITY'
60EC    86 77       STX $77       Tastennummer setzen
60EE    20 5C 79    JSR $795C     Test auf Komma
60F1    20 7B 87    JSR $877B     FRMEVL + FRESTR
60F4    A8          TAY           Länge setzen
60F5    A9 01       LDA #$01
60F7    85 26       STA $26
60F9    A9 24       LDA #$24
60FB    A6 77       LDX $77
60FD    E8          INX
60FE    20 45 A8    JSR $A845     ROMs einschalten
6101    20 65 FF    JSR $FF65     Funktionstaste definieren
6104    B0 01       BCS $6107     Wenn Fehler, Skip
6106    60          RTS
6107    4C 3A 4D    JMP $4D3A     Fehler ausgeben

******************************** BASIC-Befehl KEY

610A    F0 15       BEQ $6121     Wenn Trennzeichen, Skip
610C    C9 91       CMP #$91      Token für ON ?
610E    F0 0E       BEQ $611E     Ja, Skip
6110    C9 FE       CMP #$FE      Sondertoken 1 ?
6112    D0 CD       BNE $60E1     Nein, Skip
6114    20 80 03    JSR $0380     CHRGET
6117    C9 24       CMP #$24      Token für OFF ?
6119    F0 03       BEQ $611E     Ja, Skip
611B    4C 6C 79    JMP $796C     'SYNTAX'
611E    4C 46 48    JMP $4846     'UNIMPLEMENTED COMMAND'
6121    A2 00       LDX #$00      KEY-Nummer auf 0
6123    A0 00       LDY #$00
6125    E8          INX
6126    BD FF 0F    LDA $0FFF,X   Länge des Strings = 0 ?
6129    F0 53       BEQ $617E     Ja, Skip
612B    85 78       STA $78       Länge setzen
612D    86 77       STX $77       Nummer setzen
612F    A2 05       LDX #$05      'KEY nr,' ausgeben
6131    BD 2A A8    LDA $A82A,X
6134    CA          DEX
6135    D0 02       BNE $6139
6137    05 77       ORA $77       Nummer setzen
6139    20 69 92    JSR $9269     Zeichen ausgeben
613C    8A          TXA           Ende erreicht ?
613D    10 F2       BPL $6131     Nein, weiter ausgeben
613F    A2 07       LDX #$07
6141    B9 0A 10    LDA $100A,Y   Zeichen aus String holen
6144    C8          INY
```

```
6145  48        PHA           und retten
6146  86 79     STX $79
6148  A2 04     LDX #$04
614A  DD A3 61  CMP $61A3,X   Spezialzeichen ?
614D  F0 34     BEQ $6183     Ja, Skip
614F  CA        DEX
6150  D0 F8     BNE $614A
6152  A6 79     LDX $79       Funktionsstringnummer
6154  E0 08     CPX #$08      schon 8 ?
6156  90 07     BCC $615F     Nein, Skip
6158  D0 0A     BNE $6164     Wenn > 8, Skip
615A  A9 2B     LDA #$2B      '+' ausgeben
615C  20 69 92  JSR $9269
615F  A9 22     LDA #$22      '"' ausgeben
6161  20 69 92  JSR $9269
6164  68        PLA           Zeichen wieder holen
6165  20 69 92  JSR $9269     und ausgeben
6168  A2 09     LDX #$09
616A  C6 78     DEC $78       Alle Zeichen ausgegeben ?
616C  D0 D3     BNE $6141     Nein, weitermachen
616E  E0 09     CPX #$09
6170  90 05     BCC $6177
6172  A9 22     LDA #$22      '"'
6174  20 69 92  JSR $9269     ausgeben
6177  A9 8D     LDA #$8D      (SHIFT) + (CR)
6179  20 69 92  JSR $9269     ausgeben
617C  A6 77     LDX $77       Alle Strings ausgegeben ?
617E  E0 08     CPX #$08
6180  D0 A3     BNE $6125     Nein, weitermachen
6182  60        RTS

6183  A6 79     LDX $79
6185  BD 9A 61  LDA $619A,X   '+CHR$(' ausgeben
6188  20 69 92  JSR $9269
618B  CA        DEX
618C  E0 03     CPX #$03
618E  B0 F5     BCS $6185
6190  68        PLA           Zeichen wieder holen
6191  20 30 A8  JSR $A830     Bytewert ausgeben
6194  A9 29     LDA #$29      ')'
6196  20 69 92  JSR $9269     ausgeben
6199  A2 08     LDX #$08      Code für '+'-Ausgabe
619B  D0 CD     BNE $616A     und weiter ausgeben

****************************** Texte für KEY

619D  28 24 52 48 43 2B       '($RHC+' = '+CHR$('
```

```
61A5    22 0D 8D 22 1B          '"' (CR) (SHIFT-CR) '"' (ESC)

****************************** BASIC-Befehl PAINT

61A8    20 2F 9E    JSR $9E2F   Farbquelle auswerten
61AB    A2 04       LDX #$04
61AD    20 52 9E    JSR $9E52   Koordinaten holen
6AB0    20 F2 9D    JSR $9DF2   Aktuelle Koordinaten in Zielkoordinaten
61B3    20 1C 9E    JSR $9E1C   Modus-Byte auswerten und in (X)
61B6    E0 02       CPX #$02    Wert < 2 ?
61B8    90 03       BCC $61BD   Ja, Skip
61BA    4C 28 7D    JMP $7D28   'ILLEGAL QUANTITY'
61BD    8A          TXA
61BE    4A          LSR         Bit 0 in Bit 7 verschieben,
61BF    6A          ROR         also Modus-Bit setzen
61C0    85 8B       STA $8B     Zeichenmodus setzen
61C2    10 04       BPL $61C8
61C4    A5 83       LDA $83     Farbquelle = Hintergrundfarbe ?
61C6    F0 07       BEQ $61CF   Ja, Ende
61C8    20 49 9C    JSR $9C49   Koordinaten testen
61CB    B0 02       BCS $61CF   Wenn falsche Koordinaten, Ende
61CD    D0 01       BNE $61D0   Wenn richtig, Skip
61CF    60          RTS
61D0    20 EA 92    JSR $92EA   Garbage-Collection
61D3    8D 03 FF    STA $FF03   Write in Bank 0 setzen
61D6    A5 33       LDA $33     Feldende retten
61D8    85 24       STA $24
61DA    A5 34       LDA $34
61DC    85 25       STA $25
61DE    38          SEC
61DF    A5 35       LDA $35     ($1B) = Stringspeicherstart - $0300
61E1    E9 03       SBC #$03
61E3    85 1B       STA $1B
61E5    A5 36       LDA $36
61E7    E9 00       SBC #$00
61E9    85 1C       STA $1C

61EB    A2 00       LDX #$00
61ED    86 63       STX $63
61EF    86 64       STX $64
61F1    AE 33 11    LDX $1133   Y-Koordinate erniedrigen
61F4    D0 03       BNE $61F9
61F6    CE 34 11    DEC $1134
61F9    CE 33 11    DEC $1133
61FC    20 49 9C    JSR $9C49   Koordinaten testen
61FF    B0 02       BCS $6203   Wenn falsche Koordinaten, Skip
6201    D0 EE       BNE $61F1   Wenn frei, weiter testen
6203    EE 33 11    INC $1133   Y-Koordinate wieder erhöhen
```

```
6206   D0 03      BNE $620B
6208   EE 34 11   INC $1134

620B   20 19 9C   JSR $9C19     Koordinaten setzen
620E   AE 31 11   LDX $1131     X-Koordinate erniedrigen
6211   D0 03      BNE $6216
6213   CE 32 11   DEC $1132
6216   CE 31 11   DEC $1131
6219   A5 63      LDA $63
621B   20 7C 62   JSR $627C
621E   85 63      STA $63
6220   18         CLC
6221   AD 31 11   LDA $1131     X-Koordinate um 2 erhöhen
6224   69 02      ADC #$02
6226   8D 31 11   STA $1131
6229   90 03      BCC $622E
622B   EE 32 11   INC $1132
622E   A5 64      LDA $64
6230   20 7C 62   JSR $627C
6233   85 64      STA $64
6235   AE 31 11   LDX $1131     X-Koordinate wieder erniedrigen
6238   D0 03      BNE $623D
623A   CE 32 11   DEC $1132
623D   CE 31 11   DEC $1131
6240   EE 33 11   INC $1133     Y-Koordinate erhöhen
6243   D0 03      BNE $6248
6245   EE 34 11   INC $1134
6248   20 49 9C   JSR $9C49     Koordinaten testen
624B   B0 02      BCS $624F     Wenn falsche Koordinaten, Skip
624D   D0 BC      BNE $620B     Wenn frei, weiter füllen
624F   A2 03      LDX #$03
6251   A0 00      LDY #$00
6253   A5 25      LDA $25       PAINT-Stapel leer ?
6255   C5 34      CMP $34
6257   D0 06      BNE $625F     Nein, Skip
6259   A5 24      LDA $24
625B   C5 33      CMP $33
625D   F0 1A      BEQ $6279     Ja, Ende
625F   A5 24      LDA $24       Stapelzeiger erniedrigen
6261   D0 02      BNE $6265
6263   C6 25      DEC $25
6265   C6 24      DEC $24
6267   20 B7 03   JSR $03B7     Koordinaten vom Stapel holen
626A   8D 03 FF   STA $FF03     Write in Bank 0 setzen
626D   9D 31 11   STA $1131,X   Koordinaten setzen
6270   CA         DEX
6271   10 EC      BPL $625F
6273   20 B5 4B   JSR $4BB5     STOP-Taste prüfen
```

```
6276   4C EB 61   JMP $61EB    Zum PAINT-Start
6279   4C F2 9D   JMP $9DF2    Aktuelle Koordinaten in Zielkoordinaten

****************************** Koordinaten für PAINT retten, falls nötig

627C   48         PHA
627D   20 49 9C   JSR $9C49    Koordinaten testen
6280   B0 18      BCS $629A    Wenn falsche Koordinaten, Ende
6282   F0 16      BEQ $629A    Wenn gesetzt, Ende
6284   68         PLA          (A) wieder holen
6285   D0 16      BNE $629D    Wenn gesetzt, Ende
6287   AA         TAX
6288   A8         TAY
6289   A5 25      LDA $25      PAINT-Stack voll ?
628B   C5 1C      CMP $1C
628D   90 0F      BCC $629E    Nein, Skip
628F   D0 06      BNE $6297
6291   A5 24      LDA $24
6293   C5 1B      CMP $1B
6295   90 07      BCC $629E    Nein, Skip
6297   4C 3A 4D   JMP $4D3A    'OUT OF MEMORY'
629A   68         PLA
629B   A9 00      LDA #$00     Flag auf 0
629D   60         RTS
629E   BD 31 11   LDA $1131,X  Koordinaten auf PAINT-Stack
62A1   8D 04 FF   STA $FF04    Write in Bank 1 setzen
62A4   91 24      STA ($24),Y
62A6   8D 03 FF   STA $FF03    Read aus Bank 0 setzen
62A9   E6 24      INC $24
62AB   D0 02      BNE $62AF
62AD   E6 25      INC $25
62AF   E8         INX
62B0   E0 04      CPX #$04
62B2   D0 EA      BNE $629E
62B4   A9 80      LDA #$80     Flag für gerettet setzen
62B6   60         RTS

****************************** BASIC-Befehl BOX

62B7   20 2F 9E   JSR $9E2F    Farbquelle lesen
62BA   A2 1F      LDX #$1F
62BC   20 6D 9E   JSR $9E6D    Koordinate 1 auswerten
62BF   A2 2B      LDX #$2B
62C1   20 52 9E   JSR $9E52    Koordinate 2 auswerten
62C4   20 06 9E   JSR $9E06    Rotationswinkel auswerten
62C7   8C 54 11   STY $1154    und Winkelwert setzen
62CA   8D 55 11   STA $1155
62CD   20 1C 9E   JSR $9E1C    Ausmalflag lesen
```

```
62D0   E0 02      CPX #$02      > 1 ?
62D2   90 03      BCC $62D7     Nein, Skip
62D4   4C 28 7D   JMP $7D28     'ILLEGAL QUANTITY'
62D7   8E 6C 11   STX $116C     Ausmalflag setzen
62DA   8A         TXA
62DB   48         PHA           Ausmalwert retten
62DC   20 89 63   JSR $6389     Zielrichtung berechnen
62DF   68         PLA           Ausmalen ?
62E0   D0 1C      BNE $62FE     Ja, Skip
62E2   F0 03      BEQ $62E7     Nein, Skip

62E4   20 0B 64   JSR $640B     Zielrichtung berechnen
62E7   20 30 9B   JSR $9B30     Linie zur Zielkoordinate zeichnen
62EA   AD 4E 11   LDA $114E     Ende ?
62ED   D0 F5      BNE $62E4     Nein, weitermachen
62EF   A2 04      LDX #$04
62F1   BD 5B 11   LDA $115B,X   Aktuelle Koordinaten setzen
62F4   9D 30 11   STA $1130,X
62F7   CA         DEX
62F8   D0 F7      BNE $62F1
62FA   8E 6C 11   STX $116C     und Füllflag löschen
62FD   60         RTS

62FE   A2 00      LDX #$00      Offset auf Seitenlänge
6300   AD 49 11   LDA $1149     Seite richtig ?
6303   4A         LSR
6304   90 02      BCC $6308     Ja, Skip
6306   A2 02      LDX #$02
6308   BD 60 11   LDA $1160,X   Seitenlänge passend setzen
630B   8D 5A 11   STA $115A
630E   BD 61 11   LDA $1161,X
6311   8D 5B 11   STA $115B
6314   A9 00      LDA #$00
6316   A2 03      LDX #$03
6318   9D 56 11   STA $1156,X   Zähler löschen
631B   CA         DEX
631C   10 FA      BPL $6318
631E   A2 07      LDX #$07
6320   BD 31 11   LDA $1131,X   Koordinaten retten
6323   48         PHA
6324   CA         DEX
6325   10 F9      BPL $6320
6327   20 30 9B   JSR $9B30     Linie ziehen
632A   A2 00      LDX #$00
632C   68         PLA
632D   9D 31 11   STA $1131,X   und Koordinaten wieder setzen
6330   E8         INX
6331   E0 08      CPX #$08
```

```
6333  D0 F7     BNE $632C
6335  AD 5A 11  LDA $115A    Längenzähler schon 0 ?
6338  D0 05     BNE $633F    Nein, Skip
633A  CE 5B 11  DEC $115B
633D  30 B0     BMI $62EF    Ja, Ende
633F  CE 5A 11  DEC $115A    Längenzähler erniedrigen
6342  A2 25     LDX #$25     Offset auf Zähler X
6344  A0 1B     LDY #$1B     Offset auf Kosinuswert
6346  AD 49 11  LDA $1149    Seitennummer richtig ?
6349  4A        LSR
634A  90 02     BCC $634E    Ja, Skip
634C  A0 19     LDY #$19     Offset auf Sinuswert
634E  A9 00     LDA #$00     Übertragflag auf 0
6350  4A        LSR
6351  48        PHA
6352  20 6D 9D  JSR $9D6D    Winkelwert aufaddieren
6355  9D 31 11  STA $1131,X  und Koordinate setzen
6358  98        TYA
6359  9D 32 11  STA $1132,X
635C  68        PLA          Übertragflag holen
635D  90 02     BCC $6361    Wenn bei Rechnung kein Übertrag, Skip
635F  09 A0     ORA #$A0     Übertragflag setzen
6361  E8        INX          Offset erhöhen
6362  E8        INX
6363  A0 19     LDY #$19     Offset auf Sinuswert
6365  4E 49 11  LSR $1149    Seitennummer richtig ?
6368  90 02     BCC $636C    Ja, Skip
636A  A0 1B     LDY #$1B     Offset auf Kosinuswert
636C  2E 49 11  ROL $1149    Seitennummer korrigieren
636F  E0 27     CPX #$27     X un Y verarbeitet ?
6371  F0 DD     BEQ $6350    Nein, nochmal
6373  A2 06     LDX #$06
6375  0A        ASL          Noch Überträge ?
6376  F0 BD     BEQ $6335    Nein, weitermachen
6378  90 08     BCC $6382    Wenn kein aktueller Übertrag, Skip
637A  FE 31 11  INC $1131,X  Koordinate korrigieren
637D  D0 03     BNE $6382
637F  FE 32 11  INC $1132,X
6382  0A        ASL
6383  CA        DEX
6384  CA        DEX          Alle korrigiert ?
6385  10 F1     BPL $6378    Nein, nochmal
6387  30 95     BMI $631E    Unbedingter Sprung

****************************** Zielrichtung berechnen

6389  A0 23     LDY #$23     Offset auf Rotationswinkel
638B  20 74 9A  JSR $9A74    Winkelwerte berechnen
```

```
638E   A2 1F      LDX #$1F      Offset auf X-Wert 1
6390   A0 2B      LDY #$2B      Offset auf X-Wert 2
6392   98         TYA           Offset retten
6393   48         PHA
6394   20 99 9D   JSR $9D99     (A)/(Y) = ABS(X-Wert 2 - X-Wert 1)
6397   9D 35 11   STA $1135,X   Ergebnis setzen
639A   9D 39 11   STA $1139,X
639D   9D 41 11   STA $1141,X
63A0   98         TYA
63A1   9D 36 11   STA $1136,X
63A4   9D 3A 11   STA $113A,X
63A7   9D 42 11   STA $1142,X
63AA   68         PLA
63AB   A8         TAY           Offset auf X-Wert 2 wieder holen
63AC   20 6D 9D   JSR $9D6D     (A)/(Y) = (X-Wert 2 + X-Wert 1)
63AF   9D 31 11   STA $1131,X   Ergebnis setzen
63B2   98         TYA
63B3   9D 32 11   STA $1132,X
63B6   A0 2D      LDY #$2D      Offset auf Y-Wert 2 setzen
63B8   E8         INX           Offset auf Y-Wert 1 erhöhen
63B9   E8         INX
63BA   E0 21      CPX #$21      X und Y verarbeitet ?
63BC   F0 D4      BEQ $6392     Nein, nochmal
63BE   A9 90      LDA #$90
63C0   20 F3 9A   JSR $9AF3     Abstandswerte berechnen
63C3   AD 49 11   LDA $1149     Winkelquadrant
63C6   29 03      AND #$03      normalisieren
63C8   8D 49 11   STA $1149     und
63CB   AA         TAX           in Offset umrechnen
63CC   BD ED 63   LDA $63ED,X   und Bitmuster für Rechnung holen
63CF   20 0B 64   JSR $640B     Endgültige Zielrichtung berechnen
63D2   20 F2 9D   JSR $9DF2     und Koordinate in Zielkoordinate
63D5   AD 4E 11   LDA $114E
63D8   20 0B 64   JSR $640B     Zielrichtung berechnen
63DB   AE 49 11   LDX $1149     und Bitmuster setzen
63DE   BD ED 63   LDA $63ED,X
63E1   29 F0      AND #$F0
63E3   8D 4F 11   STA $114F
63E6   BD F1 63   LDA $63F1,X
63E9   8D 4E 11   STA $114E
63EC   60         RTS

******************************* Bitmuster
                                für Zielrichtngsberechnung

63ED   BE E4 41 1B
63F1   41 1B BE E4
```

```
****************************** Namen der Programmierer !

63F5   46 52 45 44 20 42 0D     'FRED B' (CR)
63FC   54 45 52 52 59 20 52 0D 'TERRY R' (CR)
6404   4D 49 4B 45 20 49 0D     'MIKE I' (CR)

****************************** Zielrichtungsberechnung

640B   20 67 67    JSR $6767    Zielrichtung berechnen
640E   A2 04       LDX #$04
6410   BD 32 11    LDA $1132,X
6413   0A          ASL          Vorzeichen retten
6414   7E 32 11    ROR $1132,X  und Richtung durch 2 teilen
6417   7E 31 11    ROR $1131,X
641A   90 08       BCC $6424    Wenn gerader Wert, Skip
641C   FE 31 11    INC $1131,X  Wert erhöhen
641F   D0 03       BNE $6424
6421   FE 32 11    INC $1132,X
6424   E8          INX
6425   E8          INX
6426   E0 06       CPX #$06     X und Y verarbeitet ?
6428   F0 E6       BEQ $6410    Nein, nochmal
642A   60          RTS

****************************** BASIC-Befehl SSHAPE

642B   20 74 A0    JSR $A074    Test auf HIRES-Bereich
642E   20 AF 7A    JSR $7AAF    Variable lesen
6431   8D 03 FF    STA $FF03    Write in Bank 0 setzen
6434   8D 5F 11    STA $115F    Stringadresse setzen
6437   8C 60 11    STY $1160
643A   24 0F       BIT $0F      String ?
643C   30 03       BMI $6441    Ja, Skip
643E   4C E7 77    JMP $77E7    'TYPE MISMATCH'
6441   A2 28       LDX #$28
6443   20 6D 9E    JSR $9E6D    Koordinate 1 holen
6446   A2 04       LDX #$04
6448   20 52 9E    JSR $9E52    Koordinate 2 holen
644B   A2 2A       LDX #$2A
644D   A0 06       LDY #$06
644F   A9 02       LDA #$02     Koordinatennummer
6451   85 8E       STA $8E      auf 2 setzen
6453   20 99 9D    JSR $9D99    (A)/(Y) = ABS(Koordinate 1 - Koordinate
2)
6456   AA          TAX
6457   98          TYA
6458   48          PHA
6459   A4 8E       LDY $8E      Koordinatennummer lesen
```

```
645B   20 F9 9D   JSR $9DF9     Koordinate kopieren
645E   90 0C      BCC $646C
6460   B9 59 11   LDA $1159,Y   Koordinaten setzen
6463   99 31 11   STA $1131,Y
6466   B9 5A 11   LDA $115A,Y
6469   99 32 11   STA $1132,Y
646C   8A         TXA
646D   99 59 11   STA $1159,Y
6470   99 DB 03   STA $03DB,Y
6473   68         PLA
6474   99 5A 11   STA $115A,Y
6477   99 DC 03   STA $03DC,Y
647A   A2 28      LDX #$28      Neue Koordinatenoffsets
647C   A0 04      LDY #$04      laden
647E   C6 8E      DEC $8E       Koordinatennummer erniedrigen
6480   C6 8E      DEC $8E       Beide Koordinaten verarbeitet ?
6482   F0 CF      BEQ $6453     Nein, weitermachen
6484   A0 FF      LDY #$FF
6486   8C 55 11   STY $1155
6489   AD 31 11   LDA $1131
648C   8D 5D 11   STA $115D
648F   AD 32 11   LDA $1132
6492   8D 5E 11   STA $115E
6495   98         TYA
6496   20 90 86   JSR $8690     Stringzeiger setzen
6499   8D 03 FF   STA $FF03     Write in Bank 0 setzen
649C   20 E3 9C   JSR $9CE3     Schirmadresse berechnen
649F   B1 8C      LDA ($8C),Y   Wert aus Schirm holen
64A1   90 0E      BCC $64B1     Wenn kein Fehler, Skip
64A3   AD 31 11   LDA $1131
64A6   24 D8      BIT $D8       Multicolormodus ?
64A8   10 02      BPL $64AC     Nein, Skip
64AA   38         SEC           Multicoloranpassung
64AB   2A         ROL
64AC   29 07      AND #$07
64AE   AA         TAX
64AF   A9 00      LDA #$00
64B1   24 D8      BIT $D8       Multicolormodus ?
64B3   10 01      BPL $64B6     Nein, Skip
64B5   CA         DEX
64B6   8E 61 11   STX $1161     Bitzeiger setzen
64B9   0A         ASL
64BA   CA         DEX
64BB   10 FC      BPL $64B9
64BD   6A         ROR
64BE   85 8E      STA $8E
64C0   A9 08      LDA #$08      X-Offset 8 setzen
64C2   24 D8      BIT $D8       Multicolormodus ?
```

```
64C4   10 01      BPL $64C7     Nein, Skip
64C6   4A         LSR           X-Offset auf 4
64C7   18         CLC
64C8   6D 31 11   ADC $1131     X-Koordinate erhöhen
64CB   8D 31 11   STA $1131
64CE   90 03      BCC $64D3
64D0   EE 32 11   INC $1132
64D3   20 E3 9C   JSR $9CE3     Schirmadresse berechnen
64D6   A9 00      LDA #$00      Wert 0 vorbesetzen
64D8   B0 02      BCS $64DC     Wenn falsche Koordinaten, Skip
64DA   B1 8C      LDA ($8C),Y   Wert aus Graphik holen
64DC   85 8F      STA $8F       und setzen
64DE   AE 61 11   LDX $1161
64E1   4A         LSR
64E2   E8         INX
64E3   E0 08      CPX #$08
64E5   D0 FA      BNE $64E1
64E7   05 8E      ORA $8E
64E9   EE 55 11   INC $1155
64EC   AC 55 11   LDY $1155
64EF   C0 FC      CPY #$FC      Maximallänge 252 erreicht ?
64F1   90 03      BCC $64F6     Nein, Skip
64F3   4C ED A5   JMP $A5ED     'STRING TOO LONG'
64F6   8D 04 FF   STA $FF04     Write in Bank 1 setzen
64F9   91 64      STA ($64),Y   Ergebniswert in String speichern
64FB   8D 03 FF   STA $FF03     Read aus Bank 0 setzen
64FE   AE 61 11   LDX $1161
6501   AD 59 11   LDA $1159
6504   38         SEC
6505   24 D8      BIT $D8       Multicolormodus ?
6507   10 03      BPL $650C     Nein, Skip
6509   E9 04      SBC #$04
650B   2C         .BYTE $2C
650C   A9 08      SBC #$08
650E   8D 59 11   STA $1159
6511   A5 8F      LDA $8F
6513   B0 A4      BCS $64B9
6515   CE 5A 11   DEC $115A
6518   10 9F      BPL $64B9
651A   AE 5B 11   LDX $115B
651D   D0 48      BNE $6567
651F   CE 5C 11   DEC $115C
6522   10 43      BPL $6567
6524   24 D8      BIT $D8
6526   10 06      BPL $652E
6528   0E DB 03   ASL $03DB
652B   2E DC 03   ROL $03DC
652E   A2 00      LDX #$00
```

```
6530   BD DB 03   LDA $03DB,X  Länge/Breite laden
6533   C8         INY
6534   8D 04 FF   STA $FF04    Write in Bank 1 setzen
6537   91 64      STA ($64),Y  Größe setzen
6539   8D 03 FF   STA $FF03    Read aus Bank 0 setzen
653C   E8         INX
653D   E0 04      CPX #$04
653F   D0 EF      BNE $6530
6541   C8         INY
6542   8C DB 03   STY $03DB
6545   A5 64      LDA $64
6547   8D DC 03   STA $03DC
654A   A5 65      LDA $65
654C   8D DD 03   STA $03DD
654F   A9 DB      LDA #$DB     Zeiger auf $03DB setzen
6551   85 66      STA $66
6553   A9 03      LDA #$03
6555   85 67      STA $67
6557   AD 5F 11   LDA $115F    Adresse setzen
655A   85 4B      STA $4B
655C   AD 60 11   LDA $1160
655F   85 4C      STA $4C
6561   20 94 54   JSR $5494    String zuweisen
6564   4C F2 9D   JMP $9DF2    Aktuelle Koordinaten in Zielkoordinaten
6567   CE 5B 11   DEC $115B
656A   EE 33 11   INC $1133
656D   D0 03      BNE $6572
656F   EE 34 11   INC $1134
6572   AD 5D 11   LDA $115D
6575   8D 31 11   STA $1131
6578   AD 5E 11   LDA $115E
657B   8D 32 11   STA $1132
657E   AD DB 03   LDA $03DB
6581   8D 59 11   STA $1159
6584   AD DC 03   LDA $03DC
6587   8D 5A 11   STA $115A
658A   4C 9C 64   JMP $649C

******************************** BASIC-Befehl GSHAPE

658D   20 74 A0   JSR $A074    Test auf HIRES-Bereich
6590   20 7B 87   JSR $877B    FRMEVL + FRESTR, String holen
6593   8D 03 FF   STA $FF03    Write in Bank 0 setzen
6596   8D 53 11   STA $1153    Länge setzen
6599   86 26      STX $26      Adresse setzen
659B   84 27      STY $27
659D   A2 04      LDX #$04     Offset auf Koordinate
659F   20 52 9E   JSR $9E52    Koordinate holen
```

```
65A2   20 1C 9E   JSR $9E1C     Byte-Wert in (X) holen
65A5   E0 05      CPX #$05      > 4 ?
65A7   90 03      BCC $65AC     Nein, Skip
65A9   4C 28 7D   JMP $7D28     'ILLEGAL QUANTITY'
65AC   8E 54 11   STX $1154     Modus setzen
65AF   A2 03      LDX #$03      Zähler setzen
65B1   AC 53 11   LDY $1153     Länge des Strings
65B4   C0 05      CPY #$05      > 4 ?
65B6   B0 01      BCS $65B9     Ja, Skip
65B8   60         RTS           Sonst Ende
65B9   88         DEY
65BA   A9 26      LDA #$26      Zeilenzahl und Spaltenbreite setzen
65BC   20 AB 03   JSR $03AB
65BF   8D 03 FF   STA $FF03     Write in Bank 0 setzen
65C2   9D 59 11   STA $1159,X
65C5   CA         DEX
65C6   10 F1      BPL $65B9
65C8   8E 55 11   STX $1155     Stringbyte-Zeiger auf Start
65CB   20 F2 9D   JSR $9DF2     Aktuelle Koordinaten in Zielkoordinaten
65CE   AD 59 11   LDA $1159     Spaltenbreite des SHAPE setzen
65D1   8D 5D 11   STA $115D
65D4   AD 5A 11   LDA $115A
65D7   8D 5E 11   STA $115E

65DA   A9 08      LDA #$08      Bitzähler auf 8 setzen
65DC   8D 69 11   STA $1169
65DF   EE 55 11   INC $1155     Stringzeiger erhöhen
65E2   AC 55 11   LDY $1155     und als Offset verwenden
65E5   A9 26      LDA #$26
65E7   20 AB 03   JSR $03AB     Byte-Wert aus String holen
65EA   8D 03 FF   STA $FF03     Write in Bank 0 setzen
65ED   8D 57 11   STA $1157     und Wert speichern

65F0   20 49 9C   JSR $9C49     Koordinaten testen
65F3   8D 56 11   STA $1156
65F6   0E 57 11   ASL $1157
65F9   2A         ROL
65FA   CE 69 11   DEC $1169     Bitzähler korrigieren
65FD   24 D8      BIT $D8       Multicolormodus ?
65FF   10 07      BPL $6608     Nein, Skip
6601   0E 57 11   ASL $1157     MCM-Anpassung
6604   2A         ROL
6605   CE 69 11   DEC $1169
6608   AE 54 11   LDX $1154     Zeichenmodus
660B   E0 03      CPX #$03      = 3
660D   90 0C      BCC $661B     Wenn kleiner, Skip
660F   F0 05      BEQ $6616     Wenn gleich, Skip
6611   4D 56 11   EOR $1156     Modus 4, gesetzte Schirmbits invertieren
```

```
6614   B0 11      BCS $6627     Unbedingter Sprung
6616   2D 56 11   AND $1156     Modus 3
6619   B0 0C      BCS $6627     Unbedingter Sprung
661B   E0 01      CPX #$01      Zeichenmodus = 1 ?
661D   90 08      BCC $6627     Wenn Modus = 0, Bits normal abbilden
661F   F0 04      BEQ $6625     Wenn Modus 1, Bits invertiert abbilden
6621   0D 56 11   ORA $1156     Modus 2, Bits dem Schirm überlagern
6624   2C         .BYTE $2C
6625   49 FF      EOR #$FF
6627   29 03      AND #$03      Wichtige Bits ausmaskieren
6629   24 D8      BIT $D8       Multicolormodus ?
662B   30 02      BMI $662F     Ja, Skip
662D   29 01      AND #$01      Einzelnes Bit ausmaskieren (Normgraphik)
662F   85 83      STA $83       Ergebnis als Farbquelle setzen
6631   20 19 9C   JSR $9C19     Bit an Koordinaten in Schirm
6634   EE 31 11   INC $1131     X-Koordinate erhöhen
6637   D0 03      BNE $663C
6639   EE 32 11   INC $1132
663C   38         SEC
663D   AD 5D 11   LDA $115D     Spaltenbreite
6640   24 D8      BIT $D8       nach Graphikmodus
6642   10 03      BPL $6647
6644   E9 02      SBC #$02      erniedrigen
6646   2C         .BYTE $2C
6647   E9 01      SBC #$01
6649   8D 5D 11   STA $115D
664C   AD 5E 11   LDA $115E
664F   E9 00      SBC #$00
6651   8D 5E 11   STA $115E
6654   B0 2D      BCS $6683     Wenn noch nicht < 0, Skip
6656   A2 01      LDX #$01
6658   BD 59 11   LDA $1159,X   Spaltenbreite wieder setzen
665B   9D 5D 11   STA $115D,X
665E   BD 35 11   LDA $1135,X   X-Koordinate wieder setzen
6661   9D 31 11   STA $1131,X
6664   CA         DEX
6665   10 F1      BPL $6658
6667   EE 33 11   INC $1133     Y-Koordinate erhöhen
666A   D0 03      BNE $666F
666C   EE 34 11   INC $1134
666F   38         SEC           Zeilenzahl erniedrigen
6670   AD 5B 11   LDA $115B
6673   E9 01      SBC #$01
6675   8D 5B 11   STA $115B
6678   AD 5C 11   LDA $115C
667B   E9 00      SBC #$00
667D   8D 5C 11   STA $115C
6680   B0 09      BCS $668B     Wenn noch nicht < 0, weiterausgeben
```

```
6682  60        RTS
6683  AD 69 11  LDA $1169   Bitzähler = 0 ?
6686  F0 03     BEQ $668B   Ja, Skip
6688  4C F0 65  JMP $65F0   Schleifenstart 2
668B  4C DA 65  JMP $65DA   Schleifenstart 1

******************************* BASIC-Befehl CIRCLE

668E  20 2F 9E  JSR $9E2F   Farbquelle auswerten
6691  A2 1F     LDX #$1F
6693  20 52 9E  JSR $9E52   Kreiskoordinaten ab $1150 holen
6696  20 06 9E  JSR $9E06   Kreisradius X holen
6699  8C 54 11  STY $1154   und setzen
669C  8D 55 11  STA $1155
669F  20 06 9E  JSR $9E06   Kreisradius Y holen
66A2  8C 56 11  STY $1156   und setzen
66A5  8D 57 11  STA $1157
66A8  08        PHP         Status retten
66A9  A2 23     LDX #$23
66AB  20 4A 9D  JSR $9D4A   X und Y skalieren
66AE  28        PLP         Status wieder holen
66AF  B0 11     BCS $66C2   Wenn Y-Radius angegeben, Skip
66B1  AD 54 11  LDA $1154   Y-Radius = X-Radius
66B4  8D 56 11  STA $1156
66B7  AD 55 11  LDA $1155
66BA  24 D8     BIT $D8     Multicolormodus ?
66BC  10 04     BPL $66C2   Nein, Skip
66BE  0E 56 11  ASL $1156   MCM-Anpassung
66C1  2A        ROL
66C2  8D 57 11  STA $1157
66C5  20 06 9E  JSR $9E06   Startwinkel holen
66C8  8C 5C 11  STY $115C   und setzen
66CB  8D 5D 11  STA $115D
66CE  20 06 9E  JSR $9E06   Endwinkel holen
66D1  8C 5E 11  STY $115E   und setzen
66D4  8D 5F 11  STA $115F
66D7  20 06 9E  JSR $9E06   Drehwinkel holen
66DA  85 77     STA $77     und (A) und (Y) vertauschen
66DC  98        TYA
66DD  A4 77     LDY $77
66DF  20 77 9A  JSR $9A77   Winkelwerte berechnen
66E2  A2 2D     LDX #$2D    Offset auf Endwinkel
66E4  A0 2B     LDY #$2B    Offset auf Startwinkel
66E6  20 7C 9D  JSR $9D7C   (A)/(Y) = Startwinkel - Endwinkel
66E9  90 0E     BCC $66F9   Wenn gültig, Skip
66EB  A9 68     LDA #$68    Sonst 360 Grad
66ED  A0 01     LDY #$01
66EF  20 70 9D  JSR $9D70   aufaddieren
```

```
66F2   9D 31 11   STA $1131,X  Winkelwert setzen
66F5   98         TYA
66F6   9D 32 11   STA $1132,X
66F9   A2 03      LDX #$03
66FB   BD 54 11   LDA $1154,X  Radien kopieren
66FE   9D 58 11   STA $1158,X
6701   CA         DEX
6702   10 F7      BPL $66FB
6704   A9 90      LDA #$90
6706   20 F3 9A   JSR $9AF3    Abstandswerte berechnen
6709   A2 07      LDX #$07
670B   BD 54 11   LDA $1154,X  Radien kopieren
670E   9D 60 11   STA $1160,X
6711   CA         DEX
6712   10 F7      BPL $670B
6714   20 50 67   JSR $6750    Nächste Koordinate Kreisbogen berechnen
6717   20 F2 9D   JSR $9DF2    Aktuelle Koordinaten in Zielkoordinaten
671A   A2 02      LDX #$02     Segmentwinkel vorwählen
671C   20 1E 9E   JSR $9E1E    Byte-Wert in (X) holen, falls vorhanden
671F   8A         TXA          Segmentwinkel gesetzt ?
6720   D0 03      BNE $6725    Ja, Skip
6722   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'
6725   8E 20 12   STX $1220    Segmentwinkel setzen
6728   18         CLC
6729   AD 20 12   LDA $1220    Segmentwinkel auf Startwinkel aufaddieren
672C   6D 5C 11   ADC $115C
672F   8D 5C 11   STA $115C
6732   90 03      BCC $6737
6734   EE 5D 11   INC $115D
6737   A2 2D      LDX #$2D     Offset auf Startwinkel (momentaner Wert)
6739   A0 2B      LDY #$2B     Offset auf Endwinkel
673B   20 7C 9D   JSR $9D7C    (A)/(Y) = Startwinkel - Endwinkel
673E   B0 08      BCS $6748    Wenn nicht ganz ausreichend, Skip
6740   20 50 67   JSR $6750    Nächste Koordinaten Kreisbogen berechnen
6743   20 30 9B   JSR $9B30    Linie ziehen
6746   90 E1      BCC $6729    Unbedingter Sprung
6748   A0 2D      LDY #$2D     Offset auf Endwinkel
674A   20 52 67   JSR $6752    Nächste Koordinate berechnen
674D   4C 30 9B   JMP $9B30    Linie ziehen

******************************** Nächste Koordinate Kreisbogen berechnen

6750   A0 2B      LDY #$2B     Offset auf Startwinkel
6752   20 74 9A   JSR $9A74    Aktuelle Winkelwerte berechnen
6755   A2 07      LDX #$07
```

```
6757   BD 60 11   LDA $1160,X  Kreiswinkelwerte in
675A   9D 54 11   STA $1154,X  Rechenwerte bringen
675D   CA         DEX
675E   10 F7      BPL $6757
6760   A9 50      LDA #$50
6762   20 F3 9A   JSR $9AF3    Abstandswerte berechnen
6765   A9 10      LDA #$10     Vorzeichenbit
6767   8D 4E 11   STA $114E
676A   A0 1F      LDY #$1F     Offset auf Mittelpunkt X
676C   A2 23      LDX #$23     Offset auf Winkelwerte
676E   0E 4F 11   ASL $114F    Vorzeichen für Addition setzen
6771   2E 4E 11   ROL $114E
6774   20 6B 9D   JSR $9D6B    Koordinaten aufaddieren
6777   E8         INX          Offset auf Y-Wert setzen
6778   E8         INX
6779   0E 4F 11   ASL $114F    Vorzeichenbits richtig setzen
677C   2E 4E 11   ROL $114E
677F   20 67 9D   JSR $9D67    Werte aufaddieren
6782   48         PHA          und Endergebnis retten
6783   98         TYA
6784   48         PHA
6785   A0 21      LDY #$21     Offset auf Mittelpunkt Y setzen
6787   E8         INX          Offset wieder auf X-Wert setzen
6788   E8         INX
6789   E0 27      CPX #$27     X und Y verarbeitet ?
678B   F0 E1      BEQ $676E    Nein, nochmal
678D   A2 03      LDX #$03
678F   68         PLA          Ergebnis als Zielkoordinate speichern
6790   9D 35 11   STA $1135,X
6793   CA         DEX
6794   10 F9      BPL $678F
6796   60         RTS

******************************** BASIC-Befehl DRAW

6797   20 74 A0   JSR $A074    Test auf HIRES-Bereich
679A   A2 01      LDX #$01     Farbquelle auf Vordergrundfarbe setzen
679C   86 83      STX $83
679E   20 86 03   JSR $0386    CHRGOT
67A1   C9 A4      CMP #$A4     Token für TO ?
67A3   F0 0B      BEQ $67B0    Ja, Skip
67A5   20 32 9E   JSR $9E32    Farbquelle auswerten
67A8   20 86 03   JSR $0386    CHRGOT
67AB   D0 03      BNE $67B0    Wenn kein Trennzeichen, Skip
67AD   4C FB 9B   JMP $9BFB    Koordinaten setzen
67B0   20 86 03   JSR $0386    CHRGOT
67B3   C9 2C      CMP #$2C     ',' ?
67B5   F0 05      BEQ $67BC    Ja, Skip
```

```
67B7   C9 A4      CMP #$A4    Token für TO ?
67B9   F0 01      BEQ $67BC   Ja, Skip
67BB   60         RTS
67BC   48         PHA         Zeichen retten
67BD   20 80 03   JSR $0380   CHRGET
67C0   A2 04      LDX #$04
67C2   20 70 9E   JSR $9E70   Koordinaten lesen
67C5   68         PLA         Zeichen wieder holen
67C6   10 06      BPL $67CE   Wenn nicht TO, Skip
67C8   20 30 9B   JSR $9B30
67CB   4C B0 67   JMP $67B0   Zum Schleifenstart
67CE   20 F2 9D   JSR $9DF2   Aktuelle Koordinaten in Zielkoordinaten
67D1   20 FB 9B   JSR $9BFB   Koordinaten setzen
67D4   4C B0 67   JMP $67B0   Zum Schleifenstart

******************************** BASIC-Befehl CHAR

67D7   20 32 9E   JSR $9E32   Farbquelle auswerten
67DA   A2 29      LDX #$29    40 Spalten
67DC   A0 1A      LDY #$1A    25 Zeilen
67DE   A5 D8      LDA $D8     Graphikmodus ?
67E0   D0 05      BNE $67E7   Ja, Skip
67E2   20 ED FF   JSR $FFED   Spalten und Zeilen des Textfensters holen
67E5   E8         INX         und jeweils um 1 erhöhen
67E6   C8         INY
67E7   8E 5E 11   STX $115E   Spalten- und Zeilenwerte setzen
67EA   8C 5F 11   STY $115F
67ED   20 09 88   JSR $8809   Spaltennummer auswerten
67F0   EC 5E 11   CPX $115E   > Maximalzahl ?
67F3   B0 0B      BCS $6800   Ja, Fehler
67F5   8E 5E 11   STX $115E   Spaltennummer setzen
67F8   20 09 88   JSR $8809   Zeilennummer auswerten
67FB   EC 5F 11   CPX $115F   > Maximalzahl ?
67FE   90 03      BCC $6803   Nein, Skip
6800   4C 28 7D   JMP $7D28   'ILLEGAL QUANTITY'
6803   8E 5F 11   STX $115F   Zeilennummer setzen
6806   20 86 03   JSR $0386   CHRGOT
6809   D0 04      BNE $680F   Wenn kein Trennzeichen, Skip
680B   A9 00      LDA #$00    Länge des Textes auf 0
680D   F0 06      BEQ $6815   Unbedingter Sprung
680F   20 5C 79   JSR $795C   Textstring auswerten
6812   20 7B 87   JSR $877B
6815   8D 03 FF   STA $FF03   Write in Bank 0 setzen
6818   8D 6E 11   STA $116E   Länge setzen
681B   98         TYA         Adresse retten
681C   48         PHA
681D   8A         TXA
681E   48         PHA
```

```
681F   20 1C 9E   JSR $9E1C   Inversbyte auswerten
6822   8A         TXA
6823   6A         ROR
6824   6E 3D 11   ROR $113D   und Bit 7 in $113D entsprechend setzen
6827   68         PLA         Adresse wieder holen
6828   85 24      STA $24     und setzen
682A   68         PLA
682B   85 25      STA $25
682D   A5 D8      LDA $D8     Graphikmodus ?
682F   D0 32      BNE $6863   Ja, Skip
6831   AE 5F 11   LDX $115F   Spalte und Zeile laden
6834   AC 5E 11   LDY $115E
6837   18         CLC         Flag für Setzen laden
6838   20 8D 92   JSR $928D   und Cursorposition setzen
683B   A0 00      LDY #$00    Offset auf 0
683D   2C 3D 11   BIT $113D   Invers ausgeben ?
6840   10 05      BPL $6847   Nein, Skip
6842   A9 12      LDA #$12    Code für (RVS)
6844   20 0C C0   JSR $C00C   ausgeben
6847   CC 6E 11   CPY $116E   Textende erreicht ?
684A   F0 0C      BEQ $6858   Ja, Skip
684C   20 B7 03   JSR $03B7   Zeichen aus String holen
684F   20 45 A8   JSR $A845   ROMs einschalten
6852   20 0C C0   JSR $C00C   Zeichen ausgeben
6855   C8         INY         Offset erhöhen
6856   D0 EF      BNE $6847   und weitermachen
6858   2C 3D 11   BIT $113D   Invers ausgeben ?
685B   10 05      BPL $6862   Nein, Skip
685D   A9 92      LDA #$92    Code für (OFF)
685F   20 0C C0   JSR $C00C   Zeichen ausgeben
6862   60         RTS

6863   20 74 A0   JSR $A074   Test auf HIRES-Bereich
6866   AD EC 11   LDA $11EC   Graphikzeichensatz anwählen
6869   8D 68 11   STA $1168
686C   A5 86      LDA $86     Vordergrundfarbe laden
686E   AA         TAX         in (X)
686F   48         PHA         und retten
6870   A5 83      LDA $83     Farbquelle retten
6872   48         PHA
6873   24 D8      BIT $D8     Multicolormodus ?
6875   10 0E      BPL $6885   Nein, Skip
6877   68         PLA         Farbquelle wieder holen
6878   F0 16      BEQ $6890   Wenn Hintergrundfarbe, Skip
687A   4A         LSR         Zusatzfarbe ?
687B   F0 13      BEQ $6890   Nein, Skip
687D   A6 84      LDX $84     Passende Zusatzfarbe laden
687F   90 0F      BCC $6890
```

```
6881   A6 85      LDX $85
6883   B0 0B      BCS $6890    Unbedingter Sprung
6885   A6 86      LDX $86      Vordergrundfarbe
6887   68         PLA          Farbquelle = Hintergrundfarbe ?
6888   D0 06      BNE $6890    Nein, Skip
688A   20 45 A8   JSR $A845    ROMs einschalten
688D   AE 21 D0   LDX $D021    Hintergrundfarbe aus VIC laden
6890   86 86      STX $86      und setzen
6892   AE 5F 11   LDX $115F    Zeilennummer laden
6895   A0 00      LDY #$00
6897   8C 60 11   STY $1160    Zeichenzähler auf 0
689A   AC 60 11   LDY $1160    Offset laden
689D   EE 60 11   INC $1160    Zeichenzähler erhöhen
68A0   20 B7 03   JSR $03B7    Zeichen aus String holen
68A3   8D 03 FF   STA $FF03    Write in Bank 0 setzen
68A6   CE 6E 11   DEC $116E    Länge des Strings erniedrigen
68A9   30 2C      BMI $68D7    Wenn Ende, Skip
                               (Fehler bei Längen über 127!)
68AB   C9 0E      CMP #$0E     Code für (TEXT) ?
68AD   D0 05      BNE $68B4    Nein, Skip
68AF   AD EB 11   LDA $11EB    Textzeichensatz anwählen
68B2   D0 07      BNE $68BB    Unbedingter Sprung
68B4   C9 8E      CMP #$8E     Code für (GRAPHIK) ?
68B6   D0 08      BNE $68C0    Nein, Skip
68B8   AD EC 11   LDA $11EC    Graphikzeichensatz anwählen
68BB   8D 68 11   STA $1168
68BE   D0 09      BNE $68C9    Unbedingter Sprung
68C0   AC 5E 11   LDY $115E    Spaltennummer laden
68C3   20 DB 68   JSR $68DB    Zeichen ausgeben
68C6   EE 5E 11   INC $115E    Spalte erhöhen
68C9   C0 27      CPY #$27     40 erreicht ?
68CB   90 CD      BCC $689A    Nein, zum Schleifenanfang
68CD   A0 00      LDY #$00     Spalte auf 0
68CF   8C 5E 11   STY $115E
68D2   E8         INX          Zeile erhöhen
68D3   E0 18      CPX #$18     25 erreicht ?
68D5   90 C3      BCC $689A    Nein, zum Schleifenanfang
68D7   68         PLA          Vordergrundfarbe wieder holen
68D8   85 86      STA $86
68DA   60         RTS

****************************** Zeichen an (X)/(Y) in HIRES-Schirm

68DB   48         PHA          Zeichen retten
68DC   20 70 9C   JSR $9C70    Farbwert in HIRES-Schirm
68DF   98         TYA
68E0   18         CLC
68E1   7D 33 C0   ADC $C033,X  Schirmadresse setzen
```

```
68E4   85 8C      STA $8C
68E6   BD 4C C0   LDA $C04C,X
68E9   69 00      ADC #$00       HIRES-Adresse=8*Textadresse,
68EB   06 8C      ASL $8C        da pro Zeichenposition 8 Byte
68ED   2A         ROL            benötigt werden
68EE   06 8C      ASL $8C
68F0   2A         ROL
68F1   06 8C      ASL $8C
68F3   2A         ROL
68F4   85 8D      STA $8D
68F6   8D 03 FF   STA $FF03      Write in Bank 0 setzen
68F9   A9 00      LDA #$00       Zeichengeneratoradresse berechnen
68FB   85 77      STA $77
68FD   68         PLA            Zeichenwert wieder holen
68FE   48         PHA            noch einmal retten
68FF   0A         ASL            und mit 8 multiplizieren,
6900   26 77      ROL $77        da ein Zeichen 8 Byte braucht
6902   0A         ASL
6903   0A         ASL
6904   26 77      ROL $77
6906   85 26      STA $26        und setzen
6908   A5 77      LDA $77
690A   6D 68 11   ADC $1168      Zeichengeneratoradresse aufaddieren
690D   85 27      STA $27
690F   98         TYA
6910   48         PHA
6911   A0 07      LDY #$07       Zähler setzen
6913   AD 3D 11   LDA $113D      Inversbit in Carry
6916   0A         ASL
6917   B1 26      LDA ($26),Y    Wert aus Zeichengenerator holen
6919   90 02      BCC $691D      Wenn normal, Skip
691B   49 FF      EOR #$FF       invertieren
691D   24 D8      BIT $D8        Multicolormodus ?
691F   10 2B      BPL $694C      Nein, Skip
6921   29 AA      AND #$AA       MCM-Anpassung
6923   85 77      STA $77        Wert retten
6925   A5 83      LDA $83        Falls nötig, Zusatzfarbenanpassung
6927   D0 0F      BNE $6938
6929   A5 77      LDA $77
692B   B0 07      BCS $6934
692D   4A         LSR
692E   45 77      EOR $77
6930   49 AA      EOR #$AA
6932   D0 18      BNE $694C      Unbedingter Sprung
6934   09 55      ORA #$55
6936   D0 14      BNE $694C      Unbedingter Sprung
6938   C9 02      CMP #$02       Mehrfarbmodus 1 ?
693A   D0 04      BNE $6940      Nein, Skip
```

```
693C   A5 77      LDA $77
693E   B0 0C      BCS $694C
6940   90 07      BCC $6949
6942   A5 77      LDA $77
6944   4A         LSR
6945   45 77      EOR $77
6947   90 03      BCC $694C
6949   A5 77      LDA $77
694B   4A         LSR
694C   91 8C      STA ($8C),Y  Zeichen in HIRES-Schirm
694E   88         DEY          Zähler erniedrigen
694F   10 C2      BPL $6913    Falls nötig, weitermachen
6951   68         PLA          (Y) wieder holen
6952   A8         TAY
6953   68         PLA          Zeichenwert wieder holen
6954   60         RTS

******************************* BASIC-Befehl LOCATE

6955   20 74 A0   JSR $A074    Test auf HIRES-Bereich
6958   A2 04      LDX #$04     Offset auf Koordinaten setzen
695A   20 70 9E   JSR $9E70    Koordinaten lesen
695D   4C F2 9D   JMP $9DF2    und setzen

******************************* BASIC-Befehl SCALE

6960   20 F4 87   JSR $87F4    Byte-Wert holen
6963   E0 02      CPX #$02     > 1 ?
6965   90 03      BCC $696A    Nein, Skip
6967   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'
696A   8E 6A 11   STX $116A    Skalierungsflag setzen
696D   20 86 03   JSR $0386    CHRGOT
6970   D0 14      BNE $6986    Wenn kein Trennzeichen, Skip
6972   A2 00      LDX #$00
6974   A9 50      LDA #$50     Skalierungswerte auf Standard setzen
6976   A0 32      LDY #$32
6978   24 D8      BIT $D8      Multicolormodus ?
697A   10 01      BPL $697D    Nein, Skip
697C   4A         LSR          X-Wert anpassen
697D   86 87      STX $87      Skalierung X auf 20480 (normal)
697F   85 88      STA $88      oder             10240 (MCM)
6981   86 89      STX $89      Skalierung Y auf 12800
6983   84 8A      STY $8A      (Skalierungen für 1023*1023!)
6985   60         RTS
6986   20 C4 69   JSR $69C4    X-Skalierungsfaktor holen
6989   A9 D8      LDA #$D8     Zeiger auf Konstante $68D9 setzen
698B   A0 69      LDY #$69
698D   20 89 8A   JSR $8A89    FAC#2 = Konstante
```

```
6990   20 4C 8B   JSR $8B4C   FAC#1 = FAC#2 / FAC#1
6993   20 15 88   JSR $8815   FAC#1 in Adressformat
6996   C9 00      CMP #$00    Wert = 0 ?
6998   D0 04      BNE $699E   Nein, Skip
699A   C0 00      CPY #$00
699C   F0 37      BEQ $69D5   Ja, Fehler
699E   48         PHA         Wert retten
699F   98         TYA
69A0   48         PHA
69A1   20 C4 69   JSR $69C4   Y-Skalierungsfaktor holen
69A4   A9 DD      LDA #$DD    Zeiger auf Konstante $69DD setzen
69A6   A0 69      LDY #$69
69A8   20 89 8A   JSR $8A89   FAC#2 = Konstante
69AB   20 4C 8B   JSR $8B4C   FAC#1 = FAC#2 / FAC#1
69AE   20 15 88   JSR $8815   FAC#1 in Adressformat
69B1   C9 00      CMP #$00    Wert = 0 ?
69B3   D0 04      BNE $69B9
69B5   C0 00      CPY #$00
69B7   F0 1C      BEQ $69D5   Ja, Fehler
69B9   84 89      STY $89     Y-Skalierung setzen
69BB   85 8A      STA $8A
69BD   68         PLA
69BE   85 87      STA $87     X-Skalierung setzen
69C0   68         PLA
69C1   85 88      STA $88
69C3   60         RTS

******************************* Skalierungsfaktor auswerten

69C4   20 5C 79   JSR $795C   Test auf Komma
69C7   20 D7 77   JSR $77D7   Faktor auswerten
69CA   A5 68      LDA $68     Wert zu groß ?
69CC   30 07      BMI $69D5   Ja, Fehler
69CE   A5 63      LDA $63
69D0   C9 90      CMP #$90
69D2   B0 01      BCS $69D5   Ja, Fehler
69D4   60         RTS
69D5   4C 28 7D   JMP $7D28   'ILLEGAL QUANTITY'

******************************* Divisionswerte für SCALE

69D8   99 1F FF 60 00         20971200
69E0   98 47 FF 38 00         13107000

******************************* BASIC-Befehl COLOR

69E2   20 F4 87   JSR $87F4   Farbquelle holen
69E5   E0 07      CPX #$07    Wert > 6 ?
```

```
69E5   E0 07      CPX #$07     Wert > 6 ?
69E7   B0 60      BCS $6A49    Ja, Fehler
69E9   86 77      STX $77      Wert retten
69EB   20 09 88   JSR $8809    Farbwert holen
69EE   CA         DEX
69EF   E0 10      CPX #$10     Wert ungültig ?
69F1   B0 56      BCS $6A49    Ja, Fehler
69F3   20 45 A8   JSR $A845    ROMs einschalten
69F6   8A         TXA          Farbwert in (A)
69F7   A6 77      LDX $77      Farbquelle
69F9   E0 01      CPX #$01     Vordergrund ?
69FB   F0 07      BEQ $6A04    Ja, Skip
69FD   B0 09      BCS $6A08    Wenn nicht Hintergrund, Skip
69FF   8D 21 D0   STA $D021    Hintergrundfarbe setzen
6A02   D0 3F      BNE $6A43    Unbedingter Sprung
6A04   85 86      STA $86      Vordergrundfarbe setzen
6A06   F0 3B      BEQ $6A43    Unbedingter Sprung
6A08   E0 03      CPX #$03     Farbquelle Mehrfarbmodus 2 ?
6A0A   F0 06      BEQ $6A12    Ja, Skip
6A0C   B0 08      BCS $6A16    Wenn nicht Mehrfarbmodus 1, Skip
6A0E   85 84      STA $84      Mehrfarbe 1 setzen
6A10   D0 31      BNE $6A43    Unbedingter Sprung
6A12   85 85      STA $85      Mehrfarbe 2 setzen
6A14   F0 2D      BEQ $6A43    Unbedingter Sprung
6A16   E0 05      CPX #$05     Farbquelle Textfarbe ?
6A18   F0 07      BEQ $6A21    Ja, Skip
6A1A   B0 16      BCS $6A32    Wenn nicht Randfarbe, Skip
6A1C   8D 20 D0   STA $D020    Randfarbe setzen
6A1F   D0 22      BNE $6A43    Unbedingter Sprung
6A21   24 D7      BIT $D7      80-Zeichen-Modus ?
6A23   10 08      BPL $6A2D    Nein, Skip
6A25   AA         TAX
6A26   A5 F1      LDA $F1      80-Zeichen Farbwert berechnen
6A28   29 F0      AND #$F0
6A2A   1D 4C 6A   ORA $6A4C,X
6A2D   85 F1      STA $F1      Farbwert setzen
6A2F   4C 43 6A   JMP $6A43    Farbwert
6A32   AA         TAX          für 80-Zeichen Hintergrund retten
6A33   A9 1A      LDA #$1A     VDC programmieren
6A35   8D 00 D6   STA $D600
6A38   AD 01 D6   LDA $D601
6A3B   29 F0      AND #$F0
6A3D   1D 4C 6A   ORA $6A4C,X
6A40   8D 01 D6   STA $D601
6A43   20 5C 6A   JSR $6A5C    Gepackte Farbwerte setzen
6A46   4C 1E 9E   JMP $9E1E    Byte-Wert holen
6A49   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'
```

```
****************************** Umrechnungstabelle für 80-Zeichen-Farben

6A4C    00 0F 08 07 0B 04 02 0D
6A54    0A 0C 09 06 01 05 03 0E

****************************** Gepackte Farbwerte setzen

6A5C    A5 86       LDA $86       Vordergrundfarbe laden
6A5E    0A          ASL           und Wert in oberes Nibble setzen
6A5F    0A          ASL
6A60    0A          ASL
6A61    0A          ASL
6A62    85 77       STA $77       Nibblewert retten
6A64    20 45 A8    JSR $A845     ROMs einschalten
6A67    AD 21 D0    LDA $D021     Hintergrundfarbe laden
6A6A    29 0F       AND #$0F      Wert isolieren
6A6C    05 77       ORA $77       und Vordergrundwert dazusetzen
6A6E    8D E2 03    STA $03E2     Vordergrund/Hintergrund setzen
6A71    A5 84       LDA $84       Multicolorfarbe mit
6A73    05 77       ORA $77       Vordergrund verknüpfen
6A75    8D E3 03    STA $03E3     Vordergrund/Multicolor setzen
6A78    60          RTS

****************************** BASIC-Befehl SCNCLR

6A79    D0 14       BNE $6A8F     Wenn Wertangabe, Skip
6A7B    20 8C 81    JSR $818C     Aktuellen Graphikmodus laden
6A7E    C9 05       CMP #$05      Nur VIC-Schirm löschen ?
6A80    90 09       BCC $6A8B     Ja, Skip
6A82    E9 05       SBC #$05      Nur 80-Zeichen Schirm ?
6A84    F0 58       BEQ $6ADE     Ja, Skip
6A86    48          PHA
6A87    20 DE 6A    JSR $6ADE     80-Zeichen Schirm löschen
6A8A    68          PLA
6A8B    AA          TAX           und VIC-Schirm
6A8C    4C 9B 6A    JMP $6A9B     löschen
6A8F    20 F4 87    JSR $87F4     Byte-Wert holen
6A92    E0 05       CPX #$05      80-Zeichen Schirm löschen ?
6A94    F0 48       BEQ $6ADE     Ja, Skip
6A96    90 03       BCC $6A9B     Nein, Skip
6A98    4C 28 7D    JMP $7D28     'ILLEGAL QUANTITY'
6A9B    8A          TXA           40-Zeichen Textschirm löschen ?
6A9C    F0 54       BEQ $6AF2     Ja, Skip
6A9E    20 74 A0    JSR $A074     Test auf HIRES-Bereich
6AA1    8A          TXA           Löschangabe retten
6AA2    48          PHA
6AA3    29 01       AND #$01      Volle Graphik löschen ?
6AA5    D0 21       BNE $6AC8     Ja, Skip
```

```
6AA7   20 F2 6A   JSR $6AF2    40-Zeichen Schirm löschen
6AAA   A5 D7      LDA $D7      40-Zeichen-modus ?
6AAC   48         PHA
6AAD   10 03      BPL $6AB2    Ja, Skip
6AAF   20 5F FF   JSR $FF5F    40-Zeichen ein
6AB2   AD 34 0A   LDA $0A34    Rasterzeilenwert holen
6AB5   38         SEC
6AB6   E9 30      SBC #$30     auf Schirmanfang umrechnen
6AB8   4A         LSR          und durch 8 teilen
6AB9   4A         LSR
6ABA   4A         LSR
6ABB   AA         TAX          ergibt Zeile
6ABC   A0 00      LDY #$00     Spalte auf 0
6ABE   18         CLC
6ABF   20 8D 92   JSR $928D    Cursor setzen = Textfenster (HOME)
6AC2   68         PLA          40-Zeichen-Modus ?
6AC3   10 03      BPL $6AC8    Ja, Skip
6AC5   20 5F FF   JSR $FF5F    80-Zeichen wieder ein
6AC8   68         PLA          Löschmodus wieder holen
6AC9   29 02      AND #$02     Mehrfarbschirm löschen ?
6ACB   F0 03      BEQ $6AD0    Nein, Skip
6ACD   20 17 6B   JSR $6B17    Farb-RAM mit MCM-Farwert füllen
6AD0   20 30 6B   JSR $6B30    HIRES-Schirm löschen
6AD3   A9 00      LDA #$00     Aktuelle Koordinaten auf 0 setzen
6AD5   A2 03      LDX #$03
6AD7   9D 31 11   STA $1131,X
6ADA   CA         DEX
6ADB   10 FA      BPL $6AD7
6ADD   60         RTS

******************************** 80-Zeichen Schirm löschen

6ADE   A5 D7      LDA $D7      80-Zeichen-Modus ?
6AE0   48         PHA
6AE1   30 03      BMI $6AE6    Ja, Skip
6AE3   20 5F FF   JSR $FF5F    80-Zeichen ein
6AE6   A9 93      LDA #$93     Code für (CLR)
6AE8   20 69 92   JSR $9269    ausgeben
6AEB   68         PLA          80-Zeichen-Modus ?
6AEC   30 03      BMI $6AF1    Ja, Skip
6AEE   20 5F FF   JSR $FF5F    40-Zeichen wieder ein
6AF1   60         RTS

******************************** 40-Zeichen Schirm löschen

6AF2   A5 D7      LDA $D7      40-Zeichen-Modus ?
6AF4   48         PHA
6AF5   10 03      BPL $6AFA    Ja, Skip
```

```
6AF7   20 5F FF   JSR $FF5F     40-Zeichen ein
6AFA   A9 93      LDA #$93      Code für (CLR)
6AFC   20 69 92   JSR $9269     ausgeben
6AFF   68         PLA           40-Zeichen-Modus ?
6B00   10 03      BPL $6B05     Ja, Skip
6B02   20 5F FF   JSR $FF5F     80-Zeichen wieder ein
6B05   60         RTS

******************************* Ab (Y) * 256 (X) Pages mit (A) füllen

6B06   84 8D      STY $8D       High-Byte Speicherseite setzen
6B08   A0 00      LDY #$00      und Low-Byte auf 0
6B0A   84 8C      STY $8C
6B0C   91 8C      STA ($8C),Y   Speicherseite mit (A) füllen
6B0E   88         DEY
6B0F   D0 FB      BNE $6B0C
6B11   E6 8D      INC $8D       High-Byte erhöhen
6B13   CA         DEX           Alle Speicherseiten gelöscht ?
6B14   D0 F6      BNE $6B0C     Nein, weitermachen
6B16   60         RTS

******************************* Farb-RAM mit Multicolorfarbe 2 füllen

6B17   20 45 A8   JSR $A845     ROMs einschalten
6B1A   78         SEI           Schirmverwaltung stoppen
6B1B   A5 01      LDA $01       Farb-RAM-auswahl retten
6B1D   48         PHA
6B1E   29 FE      AND #$FE      Farb-RAM 2 für Graphik anwählen
6B20   85 01      STA $01
6B22   A5 85      LDA $85       Farbwert laden
6B24   A0 D8      LDY #$D8      High-Byte Farb-RAM laden
6B26   A2 04      LDX #$04      4 Speicherseiten setzen
6B28   20 06 6B   JSR $6B06     Farb-RAM mit Farbe füllen
6B2B   68         PLA
6B2C   85 01      STA $01       Farb-RAM-auswahl wieder setzen
6B2E   58         CLI           Schirmverwaltung des Betriebssystems
6B2F   60         RTS           wieder erlauben

******************************* HIRES-Bereich löschen

6B30   A9 00      LDA #$00      0 als Löschwert
6B32   A0 20      LDY #$20      High-Byte HIRES-Bereich laden
6B34   A2 20      LDX #$20      32 Speicherseiten löschen
6B36   20 06 6B   JSR $6B06     HIRES-Bereich löschen
6B39   AD E2 03   LDA $03E2     Gepackte Normfarbwerte laden
6B3C   24 D8      BIT $D8       Multicolormodus ?
6B3E   10 03      BPL $6B43     Nein, Skip
6B40   AD E3 03   LDA $03E3     Gepackte MCM-Wert laden
```

```
6B43   20 45 A8   JSR $A845     ROMs einschalten
6B46   A0 1C      LDY #$1C      High-Byte auf Video-RAM (HIRES Farb-RAM)
6B48   A2 04      LDX #$04      4 Speicherseiten setzen
6B4A   20 06 6B   JSR $6B06     Video-RAM füllen
6B4D   A2 3F      LDX #$3F      Zeiger auf Sprites wieder setzen
6B4F   A0 07      LDY #$07
6B51   8A         TXA
6B52   99 F8 1F   STA $1FF8,Y
6B55   CA         DEX
6B56   88         DEY
6B57   10 F8      BPL $6B51
6B59   60         RTS

****************************** BASIC-Befehl GRAPHIC

6B5A   C9 9C      CMP #$9C      Token für CLR ?
6B5C   D0 0B      BNE $6B69     Nein, Skip
6B5E   20 22 A0   JSR $A022     HIRES-Bereich löschen
6B61   20 80 03   JSR $0380     Token für CLR überlesen
6B64   A9 00      LDA #$00      Graphikmodus löschen
6B66   85 D8      STA $D8
6B68   60         RTS
6B69   20 F4 87   JSR $87F4     Byte-Wert holen
6B6C   8A         TXA           und retten
6B6D   48         PHA
6B6E   E0 05      CPX #$05      80-Zeichen Schirm ?
6B70   F0 42      BEQ $6BB4     Ja, Skip
6B72   B0 4D      BCS $6BC1     Wenn größer, Fehler
6B74   BD C4 6B   LDA $6BC4,X   Entsprechenden Graphikmodus laden
6B77   85 D8      STA $D8       und setzen
6B79   F0 07      BEQ $6B82     Wenn 40-Zeichen Textschirm, Skip
6B7B   20 4F 9F   JSR $9F4F     HIRES-Bereich reservieren
6B7E   24 D8      BIT $D8       Geteilter Graphikschirm ?
6B80   50 07      BVC $6B89     Nein, Skip
6B82   24 D7      BIT $D7       40-Zeichen-Modus ?
6B84   10 03      BPL $6B89     Ja, Skip
6B86   20 5F FF   JSR $FF5F     40-Zeichen ein
6B89   20 1C 9E   JSR $9E1C     Löschmodus in (X) holen
6B8C   E0 02      CPX #$02      Zu groß ?
6B8E   B0 31      BCS $6BC1     Ja, Fehler
6B90   8A         TXA           Löschwert retten
6B91   48         PHA
6B92   A2 14      LDX #$14      Textzeile für Teilschirm auf 20
6B94   20 1E 9E   JSR $9E1E     Zeilenwert holen
6B97   E0 1A      CPX #$1A      > 25 ?
6B99   B0 26      BCS $6BC1     Ja, Fehler
6B9B   8A         TXA
6B9C   0A         ASL           Wert durch 8 teilen
```

```
6B9D   0A         ASL
6B9E   0A         ASL
6B9F   69 30      ADC #$30     + Schirmoffset
6BA1   8D 34 0A   STA $0A34    ergibt Rasterzeilennummer
6BA4   68         PLA          Löschmodus
6BA5   A8         TAY          in (Y)
6BA6   68         PLA          Graphikmodus
6BA7   AA         TAX          in (X)
6BA8   98         TYA          Löschen ?
6BA9   F0 03      BEQ $6BAE    Nein, Skip
6BAB   20 92 6A   JSR $6A92    SCNCLR anspringen
6BAE   A9 00      LDA #$00     Skalierung löschen
6BB0   8D 6A 11   STA $116A
6BB3   60         RTS
6BB4   24 D7      BIT $D7      80-Zeichen-Modus ?
6BB6   30 D1      BMI $6B89    Ja, Skip
6BB8   A5 D8      LDA $D8      40-Zeichenmodus setzen
6BBA   29 BF      AND #$BF
6BBC   85 D8      STA $D8
6BBE   4C 86 6B   JMP $6B86    und weitermachen
6BC1   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'

****************************** Modustabelle für GRAPHIC

6BC4   00 20 60 A0 E0

****************************** BASIC-Befehl BANK

6BC9   20 F4 87   JSR $87F4    Byte-Wert holen
6BCC   E0 10      CPX #$10     > 15 ?
6BCE   B0 04      BCS $6BD4    Ja, Fehler
6BD0   8E D5 03   STX $03D5    Bank setzen
6BD3   60         RTS
6BD4   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'

****************************** BASIC-Befehl SLEEP

6BD7   20 12 88   JSR $8812    Argument auswerten
6BDA   A2 00      LDX #$00
6BDC   78         SEI          und Uhr danach setzen
6BDD   8C 1D 0A   STY $0A1D
6BE0   8D 1E 0A   STA $0A1E
6BE3   8E 1F 0A   STX $0A1F
6BE6   20 0C 6C   JSR $6C0C    = Wert * 2
6BE9   20 16 6C   JSR $6C16    = Wert * 3
6BEC   20 09 6C   JSR $6C09    = Wert * 12
6BEF   AC 1D 0A   LDY $0A1D    Wert holen
6BF2   AD 1E 0A   LDA $0A1E
```

```
6BF5   AE 1F 0A   LDX $0A1F
6BF8   20 09 6C   JSR $6C09     = Wert * 48
6BFB   20 16 6C   JSR $6C16     = Wert * 60
6BFE   58         CLI           Interrupt wieder freigeben
6BFF   20 B5 4B   JSR $4BB5     STOP-Taste prüfen
6C02   AE 1F 0A   LDX $0A1F     SLEEP-Zeit abgelaufen ?
6C05   E8         INX
6C06   D0 F7      BNE $6BFF     Nein, weiterschlafen
6C08   60         RTS

****************************** Uhrzeit setzen für SLEEP

6C09   20 0C 6C   JSR $6C0C

6C0C   0E 1D 0A   ASL $0A1D     Uhrzeit * 2
6C0F   2E 1E 0A   ROL $0A1E
6C12   2E 1F 0A   ROL $0A1F
6C15   60         RTS

****************************** Uhrzeit setzen für SLEEP

6C16   48         PHA           (A),(X),(Y) auf Uhr aufaddieren
6C17   98         TYA
6C18   6D 1D 0A   ADC $0A1D
6C1B   8D 1D 0A   STA $0A1D
6C1E   68         PLA
6C1F   6D 1E 0A   ADC $0A1E
6C22   8D 1E 0A   STA $0A1E
6C25   8A         TXA
6C26   6D 1F 0A   ADC $0A1F
6C29   8D 1F 0A   STA $0A1F
6C2C   60         RTS

****************************** BASIC-Befehl WAIT

6C2D   20 03 88   JSR $8803     Adresse und Argument 1 holen
6C30   86 4B      STX $4B       Argument setzen
6C32   A2 00      LDX #$00
6C34   20 86 03   JSR $0386     2. Argument ?
6C37   F0 03      BEQ $6C3C     Nein, Skip
6C39   20 09 88   JSR $8809     2. Argument holen
6C3C   86 4C      STX $4C       und setzen
6C3E   A0 00      LDY #$00      Offset auf 0
6C40   AE D5 03   LDX $03D5     Bank anwählen
6C43   A9 16      LDA #$16      LDA ($16),Y
6C45   20 74 FF   JSR $FF74     Wert aus Bank holen
6C48   45 4C      EOR $4C       mit Argumenten verknüpfen
6C4A   25 4B      AND $4B
```

```
6C4C   F0 F0      BEQ $6C3E    Wenn 0, weiter warten
6C4E   60         RTS

******************************* BASIC-Befehl SPRITE

6C4F   20 BB 6C   JSR $6CBB    Spritenummer auswerten
6C52   20 1E 9E   JSR $9E1E    Aktivierungswert holen
6C55   90 05      BCC $6C5C    Wenn Angabe, Skip
6C57   A0 15      LDY #$15     Offset auf VIC-Spriteregister
6C59   20 9B 6C   JSR $6C9B    Sprite aktivieren/desaktivieren
6C5C   20 1E 9E   JSR $9E1E    Farbe auswerten
6C5F   90 0E      BCC $6C6F    Wenn keine Angabe, Skip
6C61   CA         DEX
6C62   E0 10      CPX #$10     Falscher Wert ?
6C64   B0 32      BCS $6C98    Ja, Fehler
6C66   8A         TXA          Wert in (A)
6C67   A6 77      LDX $77      Spritenummer
6C69   20 45 A8   JSR $A845    ROMs einschalten
6C6C   9D 27 D0   STA $D027,X  Farbwert setzen
6C6F   20 1E 9E   JSR $9E1E    Priorität auswerten
6C72   90 05      BC*?*a79    Wenn keine Angabe, Skip
6C74   A0 1B      LDY #$1B     Offset auf VIC-Register
6C76   20 9B 6C   JSR $6C9B    Priorität setzen/löschen
6C79   20 1E 9E   JSR $9E1E    X-Dehnung auswerten
6C7C   90 05      BCC $6C83    Wenn kein Angabe, Skip
6C7E   A0 1D      LDY #$1D     Offset auf VIC-Register
6C80   20 9B 6C   JSR $6C9B    X-Dehnung setzen
6C83   20 1E 9E   JSR $9E1E    Y-Dehnung auswerten
6C86   90 05      BCC $6C8D    Wenn keine Angabe, Skip
6C88   A0 17      LDY #$17     Offset auf VIC-Register
6C8A   20 9B 6C   JSR $6C9B    Y-Dehnung setzen
6C8D   20 1E 9E   JSR $9E1E    Modusangabe auswerten
6C90   90 05      BCC $6C97    Wenn keine Angabe, Skip
6C92   A0 1C      LDY #$1C     Offset auf VIC-Register
6C94   20 9B 6C   JSR $6C9B    Modus setzen
6C97   60         RTS
6C98   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'

******************************* Bits im VIC für Sprites programmieren

6C9B   8A         TXA          Setzen/löschen
6C9C   4A         LSR          in Carry
6C9D   D0 F9      BNE $6C98    Wenn Wert > 1, Fehler
6C9F   A6 77      LDX $77      Spritenummer laden
6CA1   BD B3 6C   LDA $6CB3,X  Bitwert laden
6CA4   20 45 A8   JSR $A845    ROMs einschalten
6CA7   19 00 D0   ORA $D000,Y  Bit setzen
6CAA   B0 03      BCS $6CAF    Wenn Wert setzen, Skip
```

```
6CAC   5D B3 6C   EOR $6CB3,X  Bit löschen
6CAF   99 00 D0   STA $D000,Y  und in VIC speichern
6CB2   60         RTS

****************************** Zweierpotenzen für SPRITE

6CB3   01 02 04 08 10 20 40 80

****************************** Spritenummer auswerten

6CBB   20 F4 87   JSR $87F4    Byte-Wert holen
6CBE   CA         DEX          und erniedrigen
6CBF   E0 08      CPX #$08     > 7 ?
6CC1   B0 D5      BCS $6C98    Ja, Fehler
6CC3   86 77      STX $77      Spritenummer setzen
6CC5   60         RTS

****************************** BASIC-Befehl MOVSPR

6CC6   20 BB 6C   JSR $6CBB    Spritenummer auswerten
6CC9   20 9E 6D   JSR $6D9E    Wertangabe auswerten
6CCC   2C 6E 11   BIT $116E    Komma angegeben ?
6CCF   50 03      BVC $6CD4    Ja, Skip
6CD1   4C 6C 79   JMP $796C    'SYNTAX'
6CD4   8C 35 11   STY $1135    X und Y-Koordinaten setzen
6CD7   8C 37 11   STY $1137
6CDA   8D 36 11   STA $1136
6CDD   8D 38 11   STA $1138
6CE0   20 9E 6D   JSR $6D9E    Weiteren Parameter auswerten
6CE3   2C 6E 11   BIT $116E    Komma angegeben ?
6CE6   50 61      BVC $6D49    Ja, Skip
6CE8   30 3A      BMI $6D24    Wenn Relativkoordinaten, Skip
6CEA   98         TYA          Geschwindigkeit
6CEB   48         PHA          retten
6CEC   A0 04      LDY #$04
6CEE   20 74 9A   JSR $9A74    Winkelwerte berechnen
6CF1   A6 77      LDX $77      Spritenummer
6CF3   BC D9 6D   LDY $6DD9,X  Offset auf Spritewerttabelle laden
6CF6   A9 00      LDA #$00     Geschwindigkeit
6CF8   99 7E 11   STA $117E,Y  auf 0 setzen
6CFB   C8         INY
6CFC   A2 03      LDX #$03     Winkelwerte durch 2 teilen
6CFE   5E 4A 11   LSR $114A,X
6D01   CA         DEX
6D02   7E 4A 11   ROR $114A,X
6D05   CA         DEX
6D06   10 F6      BPL $6CFE
6D08   E8         INX          (X) = 0
```

```
6D09   BD 49 11   LDA $1149,X  Winkelwerte
6D0C   C8         INY
6D0D   99 7E 11   STA $117E,Y  in Spritewerttabelle übertragen
6D10   E0 04      CPX #$04
6D12   D0 F4      BNE $6D08
6D14   A9 00      LDA #$00
6D16   C8         INY
6D17   99 7E 11   STA $117E,Y
6D1A   CA         DEX
6D1B   D0 F9      BNE $6D16
6D1D   68         PLA          Geschwindigkeit holen
6D1E   29 0F      AND #$0F     Anpassung auf 0 bis 15
6D20   99 74 11   STA $1174,Y  und Speichern
6D23   60         RTS

****************************** Relativkoordinaten setzen

6D24   20 39 81   JSR $8139    (X) und (Y) vertauschen
6D27   A8         TAY
6D28   8A         TXA
6D29   20 77 9A   JSR $9A77    Winkelwerte berechnen
6D2C   A2 04      LDX #$04     Offset auf Koordinate
6D2E   20 4A 9D   JSR $9D4A    Koordinaten skalieren
6D31   A2 04      LDX #$04
6D33   18         CLC
6D34   20 CE 9A   JSR $9ACE
6D37   9D 31 11   STA $1131,X  Koordinaten setzen
6D3A   98         TYA
6D3B   9D 32 11   STA $1132,X
6D3E   E8         INX
6D3F   E8         INX
6D40   E0 06      CPX #$06
6D42   F0 F0      BEQ $6D34
6D44   6E 6E 11   ROR $116E    Bit 7 setzen
6D47   30 0B      BMI $6D54    Unbedingter Sprung

****************************** Absolutkoordinaten setzen

6D49   8C 37 11   STY $1137    Y-Koordinate setzen
6D4C   8D 38 11   STA $1138
6D4F   A2 04      LDX #$04
6D51   20 4A 9D   JSR $9D4A    Koordinaten skalieren
6D54   A5 77      LDA $77      Spritenummer
6D56   AA         TAX          als Offset auf Tabelle
6D57   0A         ASL          und
6D58   A8         TAY          als Offset auf VIC-Tabelle laden
6D59   AD 37 11   LDA $1137    Y-Koordinate laden
6D5C   0E 6E 11   ASL $116E    Y = Relativkoordinaten ?
```

```
6D5F   90 09      BCC $6D6A    nein, Skip
6D61   18         CLC
6D62   10 03      BPL $6D67    Wenn positiv, Skip
6D64   49 FF      EOR #$FF     Wert invertieren
6D66   38         SEC
6D67   79 D7 11   ADC $11D7,Y  zu alter Y-Koordinate addieren
6D6A   78         SEI          IRQ sperren
6D6B   99 D7 11   STA $11D7,Y  und Y-Koordinate speichern
6D6E   AD 35 11   LDA $1135    X-Koordinate Low laden
6D71   0E 6E 11   ASL $116E    X = Relativkoordinate ?
6D74   10 12      BPL $6D88    Nein, Skip
6D76   18         CLC
6D77   79 D6 11   ADC $11D6,Y  Koordinatenwert aufaddieren
6D7A   99 D6 11   STA $11D6,Y  und speichern
6D7D   B0 03      BCS $6D82    Bei Übertrag
6D7F   EE 36 11   INC $1136    X-High erhöhen
6D82   AD E6 11   LDA $11E6    VIC X-Übertrag laden
6D85   4C 91 6D   JMP $6D91

6D88   99 D6 11   STA $11D6,Y  Koordinate speichern
6D8B   AD E6 11   LDA $11E6    und VIC X-Übertrag
6D8E   1D B3 6C   ORA $6CB3,X
6D91   4E 36 11   LSR $1136    nach X-High
6D94   B0 03      BCS $6D99
6D96   5D B3 6C   EOR $6CB3,X
6D99   8D E6 11   STA $11E6    setzen
6D9C   58         CLI
6D9D   60         RTS

******************************** Angabe für MOVSPR auswerten

6D9E   20 C6 6D   JSR $6DC6    Test auf Komma und andere Zeichen
6DA1   6E 6E 11   ROR $116E    Kommastatus setzen
6DA4   10 0B      BPL $6DB1    Wenn Komma, Skip
6DA6   C9 3B      CMP #$3B     ';' ?
6DA8   F0 13      BEQ $6DBD    Ja, Skip
6DAA   C9 23      CMP #$23     '#' ?
6DAC   F0 0E      BEQ $6DBC    Ja, Skip
6DAE   4C 6C 79   JMP $796C    'SYNTAX'
6DB1   20 86 03   JSR $0386    CHRGOT
6DB4   C9 AA      CMP #$AA     Token für + ?
6DB6   F0 05      BEQ $6DBD    Ja, Skip
6DB8   C9 AB      CMP #$AB     Token für - ?
6DBA   F0 01      BEQ $6DBD    Ja, Skip
6DBC   18         CLC          Flag für anderes Zeichen
6DBD   6E 6E 11   ROR $116E    Status bei +,-,; setzen (BEQ = SEC)
6DC0   20 D7 77   JSR $77D7    Wert holen
```

```
6DC3   4C 19 88   JMP $8819    und in Adressformat

****************************** Test auf Komma oder andere Zeichen

6DC6   20 86 03   JSR $0386    CHRGOT
6DC9   F0 0D      BEQ $6DD8    Wenn Trennzeichen, Skip
6DCB   C9 2C      CMP #$2C     ',' ?
6DCD   18         CLC          Flag für Komma
6DCE   F0 01      BEQ $6DD1    Ja, Skip
6DD0   38         SEC          Flag für anderes
6DD1   08         PHP          Flags retten
6DD2   48         PHA          Zeichen retten
6DD3   20 80 03   JSR $0380    und Zeichen überlesen
6DD6   68         PLA
6DD7   28         PLP
6DD8   60         RTS

****************************** 11er Tabellenoffsets für Spritedaten

6DD9   00 0B 16 21 2C 37 42 4D 0,11,22,33,44,55,66,77

****************************** BASIC-Befehl PLAY

6DE1   20 7B 87   JSR $877B    FRMEVL + FRESTR String auswerten
6DE4   8D 03 FF   STA $FF03    Write in Bank 0 setzen
6DE7   85 77      STA $77      Länge speichern
6DE9   20 CE 6F   JSR $6FCE    Werte zurücksetzen
6DEC   85 78      STA $78      Stringoffset auf 0
6DEE   A4 78      LDY $78      Offset laden
6DF0   C4 77      CPY $77      Stringlänge erreicht ?
6DF2   F0 0D      BEQ $6E01    Ja, Ende
6DF4   20 B7 03   JSR $03B7    Zeichen aus String holen
6DF7   8D 03 FF   STA $FF03    Write in Bank 0 setzen
6DFA   20 02 6E   JSR $6E02    Zeichen analysieren
6DFD   E6 78      INC $78      Offset erhöhen
6DFF   D0 ED      BNE $6DEE    Unbedingter Sprung zum Schleifenanfang
6E01   60         RTS

****************************** Zeichenanalyse für PLAY

6E02   C9 20      CMP #$20     Space ?
6E04   D0 01      BNE $6E07    Nein, Skip
6E06   60         RTS
6E07   C9 41      CMP #$41     < 'A' ?
6E09   90 07      BCC $6E12    Ja, Skip
6E0B   C9 48      CMP #$48     > 'G' ?
6E0D   B0 03      BCS $6E12    Ja, Skip
6E0F   4C 1E 6F   JMP $6F1E    Note auswerten
```

```
6E12   A2 04      LDX #$04
6E14   DD E7 6F   CMP $6FE7,X  Notendauerangabe 'W','H','Q','I','S' ?
6E17   D0 03      BNE $6E1C    Nein, Skip
6E19   4C 07 6F   JMP $6F07    Notendauer setzen
6E1C   CA         DEX
6E1D   10 F5      BPL $6E14
6E1F   C9 52      CMP #$52     'R' ?
6E21   D0 03      BNE $6E26    Nein, Skip
6E23   4C 78 6F   JMP $6F78    Filterresonanz festlegen
6E26   C9 2E      CMP #$2E     '.' ?
6E28   D0 03      BNE $6E2D
6E2A   4C 03 6F   JMP $6F03    Punktierte Note setzen
6E2D   A2 05      LDX #$05
6E2F   DD EC 6F   CMP $6FEC,X  Steuerung 'V','O','T','X','U','M' ?
6E32   D0 03      BNE $6E37    Nein, Skip
6E34   4C 52 6F   JMP $6F52    Musiksteuerung durchführen
6E37   CA         DEX
6E38   10 F5      BPL $6E2F
6E3A   C9 23      CMP #$23     '#' ?
6E3C   D0 03      BNE $6E41    Nein, Skip
6E3E   4C 69 6F   JMP $6F69    Halbton erhöhen
6E41   C9 24      CMP #$24     '$' ?
6E43   D0 03      BNE $6E48    Nein, Skip
6E45   4C 6C 6F   JMP $6F6C    Halbton erniedrigen (b)
6E48   38         SEC
6E49   E9 30      SBC #$30
6E4B   C9 0A      CMP #$0A     Zahl ?
6E4D   90 03      BCC $6E52    Ja, Skip
6E4F   4C FD 6E   JMP $6EFD    Fehler ausgeben
6E52   0E 26 01   ASL $0126    Stimme setzen ?
6E55   B0 46      BCS $6E9D    Ja, Skip
6E57   0E 26 01   ASL $0126    Oktave setzen ?
6E5A   B0 4C      BCS $6EA8    Ja, Skip
6E5C   0E 26 01   ASL $0126    Klang setzen ?
6E5F   B0 50      BCS $6EB1    Ja, Skip
6E61   0E 26 01   ASL $0126    Filter setzen ?
6E64   90 77      BCC $6EDD    Nein, Lautstärke setzen, Skip
6E66   C9 02      CMP #$02     Wert = 1 oder 2 ?
6E68   90 03      BCC $6E6D    Ja, Skip
6E6A   4C FD 6E   JMP $6EFD    Fehler

******************************** Filter setzen

6E6D   4A         LSR          Filterbit in Carry
6E6E   AC 2F 12   LDY $122F    Stimmennummer
6E71   BE E4 6F   LDX $6FE4,Y  Offset auf Stimmentabelle laden
6E74   BD 24 12   LDA $1224,X  Stimme fertig ?
6E77   10 FB      BPL $6E74    Nein, warten
```

```
6E79   B9 B3 6C   LDA $6CB3,Y  Filterbits
6E7C   0D 73 12   ORA $1273    setzen
6E7F   B0 03      BCS $6E84    Wenn Filter an, Skip
6E81   59 B3 6C   EOR $6CB3,Y  Filterbits wieder löschen
6E84   8D 73 12   STA $1273    und Filterwert löschen
6E87   AD 74 12   LDA $1274    Filterwert kopieren
6E8A   8D 75 12   STA $1275
6E8D   A2 03      LDX #$03     Filter programmieren
6E8F   BD 71 12   LDA $1271,X
6E92   20 45 A8   JSR $A845    ROMs einschalten
6E95   9D 15 D4   STA $D415,X  Werte in SID
6E98   CA         DEX
6E99   10 F4      BPL $6E8F
6E9B   30 5A      BMI $6EF7

****************************** Stimme setzen

6E9D   AA         TAX
6E9E   CA         DEX
6E9F   E0 03      CPX #$03     Stimmennummer gültig ?
6EA1   B0 5A      BCS $6EFD    Nein, Fehler
6EA3   8E 2F 12   STX $122F    Stimme setzen
6EA6   90 4F      BCC $6EF7    Unbedingter Sprung

****************************** Oktave setzen

6EA8   C9 07      CMP #$07     Oktave gültig ?
6EAA   B0 51      BCS $6EFD    Nein, Fehler
6EAC   8D 2B 12   STA $122B    Oktave setzen
6EAF   90 46      BCC $6EF7    Unbedingter Sprung

****************************** Klang setzen

6EB1   AA         TAX          Klangnummer als Offset
6EB2   20 45 A8   JSR $A845    ROMs einschalten
6EB5   AC 2F 12   LDY $122F    Stimmennummer
6EB8   BD 53 12   LDA $1253,X  Wellenform setzen
6EBB   99 30 12   STA $1230,Y
6EBE   B9 39 70   LDA $7039,Y  Offset auf SID-Register laden
6EC1   A8         TAY
6EC2   BD 3F 12   LDA $123F,X  Attack/Decay setzen
6EC5   99 05 D4   STA $D405,Y
6EC8   BD 49 12   LDA $1249,X  Sustain/Release setzen
6ECB   99 06 D4   STA $D406,Y
6ECE   BD 5D 12   LDA $125D,X  Pulsbreite-Low setzen
6ED1   99 02 D4   STA $D402,Y
6ED4   BD 67 12   LDA $1267,X  Pulsbreite-High setzen
6ED7   99 03 D4   STA $D403,Y
```

```
6EDA   4C F7 6E   JMP $6EF7

******************************** Lautstärke setzen

6EDD   AA         TAX          Lautstärke als Offset verwenden
6EDE   AD 74 12   LDA $1274    Filterwert nehmen
6EE1   29 F0      AND #$F0
6EE3   1D 3C 70   ORA $703C,X  mit Lautstärketabelle verknüpfen
6EE6   8D 74 12   STA $1274    und wieder setzen
6EE9   AD 75 12   LDA $1275    Filterwert nehmen
6EEC   29 F0      AND #$F0
6EEE   1D 3C 70   ORA $703C,X  mit Lautstärketabelle verknüpfen
6EF1   20 45 A8   JSR $A845    ROMs einschalten
6EF4   8D 18 D4   STA $D418    und Wert in SID setzen

******************************** Steuerungsbits löschen

6EF7   A9 00      LDA #$00
6EF9   8D 26 01   STA $0126
6EFC   60         RTS

******************************** 'ILLEGAL QUANTITY'

6EFD   20 F7 6E   JSR $6EF7    Steuerung rücksetzen
6F00   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'

******************************** Punktierung einer Note setzen

6F03   8D 33 12   STA $1233
6F06   60         RTS

******************************** Notenlänge setzen

6F07   A0 80      LDY #$80     ($1229) = $0480
6F09   8C 29 12   STY $1229
6F0C   A0 04      LDY #$04
6F0E   8C 2A 12   STY $122A
6F11   CA         DEX          Notenlänge erreicht ?
6F12   30 09      BMI $6F1D    Ja, Ende
6F14   4E 2A 12   LSR $122A    Wert durch 2 teilen
6F17   6E 29 12   ROR $1229
6F1A   4C 11 6F   JMP $6F11    und weitermachen
6F1D   60         RTS

******************************** Notenwert spielen

6F1E   38         SEC
6F1F   E9 41      SBC #$41     Von ASCII zu Offset
```

```
6F21   AA         TAX          Als Offset laden
6F22   BD F2 6F   LDA $6FF2,X  Halbtonwert für Note laden
6F25   AA         TAX          und retten
6F26   A9 06      LDA #$06     6 Oktaven
6F28   38         SEC          minus
6F29   ED 2B 12   SBC $122B    Oktavennummer
6F2C   A8         TAY          ergibt Oktavenwert
6F2D   8A         TXA          Halbtonwert wieder holen
6F2E   18         CLC
6F2F   6D 2C 12   ADC $122C    und Halbtonflag addieren (# oder b)
6F32   10 03      BPL $6F37    Wenn > 0, Skip
6F34   A9 0B      LDA #$0B     12. Halbton
6F36   C8         INY          in niedrigerer Oktave
6F37   C9 0C      CMP #$0C     Halbtonwert > 11 ?
6F39   90 03      BCC $6F3E    nein, Skip
6F3B   A9 00      LDA #$00     1. Halbton
6F3D   88         DEY          in höherer Oktave
6F3E   AA         TAX          Halbtonwert als Offset verwenden
6F3F   BD F9 6F   LDA $6FF9,X  Pitchwert-Low
6F42   8D 2D 12   STA $122D    setzen
6F45   BD 05 70   LDA $7005,X  Pitch-High laden
6F48   88         DEY          Oktave erreicht ?
6F49   30 27      BMI $6F72    Ja, Skip
6F4B   4A         LSR          Pitchwert durch 2 teilen
6F4C   6E 2D 12   ROR $122D
6F4F   4C 48 6F   JMP $6F48    und weiterschleifen

****************************** Steuerungsbefehle auswerten

6F52   C9 4D      CMP #$4D     'M' ?
6F54   F0 07      BEQ $6F5D    Ja, Skip
6F56   BD 1C 9D   LDA $9D1C,X  Steuerungsbit holen
6F59   8D 26 01   STA $0126    und setzen
6F5C   60         RTS

****************************** Auf Musikende aller Stimmen warten

6F5D   A0 05      LDY #$05
6F5F   B9 23 12   LDA $1223,Y  Stimme fertig ?
6F62   10 FB      BPL $6F5F    Nein, warten
6F64   88         DEY
6F65   88         DEY          Alle Stimmen fertig ?
6F66   10 F7      BPL $6F5F    Nein, weiterwarten
6F68   60         RTS

****************************** Halbton setzen (# oder b)

6F69   A9 01      LDA #$01     Halbton erhöhen #
```

```
6F6B   2C          .BYTE $2C
6F6C   A9 FF       LDA #$FF      Halbton erniedrigen b
6F6E   8D 2C 12    STA $122C
6F71   60          RTS

****************************** Notenwert spielen

6F72   8D 2E 12    STA $122E     Pitchwert-High setzen
6F75   A9 00       LDA #$00      Flag für Note
6F77   2C          .BYTE $2C
6F78   A9 FF       LDA #$FF      Flag für Pause
6F7A   48          PHA           Flag retten
6F7B   AE 2F 12    LDX $122F     Stimmennummer
6F7E   BC E4 6F    LDY $6FE4,X   Offset auf Stimmentabelle laden
6F81   B9 24 12    LDA $1224,Y   Stimme fertig ?
6F84   10 FB       BPL $6F81     Nein, warten
6F86   AD 29 12    LDA $1229     Notendauer setzen
6F89   99 23 12    STA $1223,Y
6F8C   AD 2A 12    LDA $122A
6F8F   99 24 12    STA $1224,Y
6F92   AD 33 12    LDA $1233     Punktierung ?
6F95   F0 17       BEQ $6FAE     Nein, Skip
6F97   AD 2A 12    LDA $122A     Notendauer halbieren
6F9A   4A          LSR
6F9B   48          PHA
6F9C   AD 29 12    LDA $1229
6F9F   6A          ROR
6FA0   18          CLC           und dazuaddieren
6FA1   79 23 12    ADC $1223,Y
6FA4   99 23 12    STA $1223,Y
6FA7   68          PLA
6FA8   79 24 12    ADC $1224,Y
6FAB   99 24 12    STA $1224,Y
6FAE   68          PLA           Note spielen ?
6FAF   30 1D       BMI $6FCE     Nein, Pause, Skip
6FB1   20 45 A8    JSR $A845     ROMs einschalten
6FB4   BC 39 70    LDY $7039,X   Stimmenoffset laden
6FB7   AD 2D 12    LDA $122D     Pitchwert eintragen
6FBA   99 00 D4    STA $D400,Y
6FBD   AD 2E 12    LDA $122E
6FC0   99 01 D4    STA $D401,Y
6FC3   A9 08       LDA #$08      SID entriegeln (TEST-Bit setzen!)
6FC5   99 04 D4    STA $D404,Y
6FC8   BD 30 12    LDA $1230,X   Wellenform programmieren
6FCB   99 04 D4    STA $D404,Y
```

```
******************************* Flags löschen

6FCE   A9 00      LDA #$00
6FD0   8D 2C 12   STA $122C    Halbtonflag löschen
6FD3   8D 33 12   STA $1233    Punktierung löschen
6FD6   60         RTS

******************************* BASIC-Befehl TEMPO

6FD7   20 F4 87   JSR $87F4    Byte-Wert holen
6FDA   8A         TXA          = 0 ?
6FDB   F0 04      BEQ $6FE1    Ja, Fehler
6FDD   8E 22 12   STX $1222    Tempo setzen
6FE0   60         RTS
6FE1   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'

******************************* Offsets für Stimmentabelle

6FE4   00 02 04

******************************* Kommandotabellen für PLAY

6FE7   57 48 51 49 53          'W','H','Q','I','S'

6FEC   56 4F 54 58 55 4D       'V','O','T','X','U','M'

******************************* Halbtonzahlen für Notenwerte

6FF2   09 0B 00 02 04 05 07

******************************* Pitchtabelle Low

6FF9   2F B6 83 99 FC B1 BD 25
7001   EF 20 BE D1

******************************* Pitchtabelle High

7005   4C 50 55 5A 5F 65 6B 72
700D   78 80 87 8F

******************************* Vorbesetzungswerte für die Hüllkurven

7011   09 C0 00 05 94 09 09 09 Attack/Decay-Werte
7019   89 09
701B   00 C0 F0 50 40 21 00 90 Sustain/Release-Werte
7023   41 00
7025   41 21 11 81 11 21 41 41 Wellenformwerte für den SID-Baustein
```

```
702D   41 11

****************************** Vorbesetzung für die Pulswellen

702F   06 00 00 00 00 00 02 08 High-Bytes der Pulswerte
7037   02 00

****************************** SID-Stimmenoffsets

7039   00 07 0E                0,7,14

****************************** Lautstärkentabelle

703C   00 01 03 05 07 08 0A 0C
7044   0E 0F

****************************** BASIC-Befehl FILTER

7046   48         PHA          Aktuelles Zeichen retten
7047   A0 03      LDY #$03     Filterwerte
7049   B9 71 12   LDA $1271,Y  mit alten Werten
704C   99 34 12   STA $1234,Y  vorbesetzen
704F   88         DEY
7050   10 F7      BPL $7049
7052   68         PLA          Zeichen wieder holen
7053   C9 2C      CMP #$2C     ',' ?
7055   F0 19      BEQ $7070    Ja, Skip
7057   20 12 88   JSR $8812    Ausdruck nach Komma holen
705A   C9 08      CMP #$08     Wert > 2047 ?
705C   B0 60      BCS $70BE    Ja, Fehler
705E   8C 34 12   STY $1234    Filterfrequenz setzen
7061   8C 35 12   STY $1235
7064   4A         LSR
7065   6E 35 12   ROR $1235
7068   4A         LSR
7069   6E 35 12   ROR $1235
706C   4A         LSR
706D   6E 35 12   ROR $1235
7070   A9 10      LDA #$10     Hilfsregister vorbesetzen
7072   8D 38 12   STA $1238
7075   20 1E 9E   JSR $9E1E    Byte-Wert holen
7078   90 17      BCC $7091    Wenn keine Angabe, Skip
707A   E0 01      CPX #$01     Wert korrekt ?
707C   90 05      BCC $7083    Ja, Skip
707E   F0 03      BEQ $7083    Ja, Skip
7080   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'
7083   AD 37 12   LDA $1237    Wert laden
7086   0D 38 12   ORA $1238    und Filterbit setzen
```

```
7089   B0 03      BCS $708E     Wenn gesetzt werden soll, Skip
708B   4D 38 12   EOR $1238     Filterbit löschen
708E   8D 37 12   STA $1237     Filterwert speichern
7091   0E 38 12   ASL $1238     Hilfsregister linksverschieben
7094   10 DF      BPL $7075     Schleife, bis alle Filterbits
abgearbeitet
7096   20 1E 9E   JSR $9E1E     Resonanz auswerten
7099   90 17      BCC $70B2     Wenn keine Angabe, Skip
709B   E0 10      CPX #$10      > 15 ?
709D   B0 1F      BCS $70BE     Ja, Fehler
709F   8A         TXA
70A0   0A         ASL           Resonanz anpassen
70A1   0A         ASL
70A2   0A         ASL
70A3   0A         ASL
70A4   8D 39 12   STA $1239
70A7   AD 36 12   LDA $1236
70AA   29 0F      AND #$0F
70AC   0D 39 12   ORA $1239     und mit Filterdaten verknüpfen
70AF   8D 36 12   STA $1236
70B2   A0 03      LDY #$03      Filterwerte wieder programmieren
70B4   B9 34 12   LDA $1234,Y
70B7   99 71 12   STA $1271,Y
70BA   88         DEY
70BB   10 F7      BPL $70B4
70BD   60         RTS
70BE   4C 28 7D   JMP $7D28     'ILLEGAL QUANTITY'

******************************* BASIC-Befehl ENVELOPE

70C1   20 F4 87   JSR $87F4     Klangnummer auswerten
70C4   E0 0A      CPX #$0A      Zu groß ?
70C6   90 03      BCC $70CB     Nein, Skip
70C8   4C 28 7D   JMP $7D28     'ILLEGAL QUANTITY'
70CB   8E 3A 12   STX $123A     Klangnummer setzen
70CE   BD 3F 12   LDA $123F,X   Klangwerte mit alten Werten vorbesetzen
70D1   8D 3B 12   STA $123B
70D4   BD 49 12   LDA $1249,X
70D7   8D 3C 12   STA $123C
70DA   BD 53 12   LDA $1253,X
70DD   8D 3D 12   STA $123D
70E0   A2 00      LDX #$00
70E2   8E 3E 12   STX $123E     Offset auf 0
70E5   20 1E 9E   JSR $9E1E     Byte-Wert holen
70E8   90 16      BCC $7100     Wenn keine Angabe, Skip
70EA   8A         TXA           Wert in (A)
70EB   0A         ASL           und oberes Nibble isolieren
70EC   0A         ASL
```

```
70ED   0A         ASL
70EE   0A         ASL
70EF   8D 39 12   STA $1239    Wert setzen
70F2   AE 3E 12   LDX $123E
70F5   BD 3B 12   LDA $123B,X
70F8   29 0F      AND #$0F     Altes unteres Nibble holen
70FA   0D 39 12   ORA $1239    und mit neuem verknüpfen
70FD   9D 3B 12   STA $123B,X  Wert wieder setzen
7100   20 1E 9E   JSR $9E1E    Byte-Wert holen
7103   90 14      BCC $7119    Wenn keine Angabe, Skip
7105   8A         TXA
7106   29 0F      AND #$0F     Unteres Nibble isolieren
7108   8D 39 12   STA $1239    Wert setzen
710B   AE 3E 12   LDX $123E
710E   BD 3B 12   LDA $123B,X
7111   29 F0      AND #$F0     Oberes Nibble isolieren
7113   0D 39 12   ORA $1239    und mit neuem Wert verknüpfen
7116   9D 3B 12   STA $123B,X  Wert wieder setzen
7119   AE 3E 12   LDX $123E    Attack/Decay
711C   E8         INX          und Sustain/Release
711D   E0 01      CPX #$01     verarbeitet ?
711F   F0 C1      BEQ $70E2    Nein, nochmal Werte holen
7121   20 1E 9E   JSR $9E1E    Wellenform holen
7124   90 10      BCC $7136    Wenn keine Angabe, Skip
7126   A9 15      LDA #$15     Wert mit Ringmodulation vorbesetzen
7128   E0 04      CPX #$04     Ringmodulation ?
712A   F0 07      BEQ $7133    Ja, Skip
712C   B0 9A      BCS $70C8    Wenn falsche Angabe, Fehler
712E   BD B7 6C   LDA $6CB7,X  Bit für entsprechende Wellenform holen
7131   09 01      ORA #$01     KEY-Bit setzen
7133   8D 3D 12   STA $123D    und Wert speichern
7136   20 06 9E   JSR $9E06    Impulsbreite in (A)/(Y) holen
7139   90 13      BCC $714E    Wenn keine Angabe, Skip
713B   AA         TAX
713C   AD 3D 12   LDA $123D
713F   29 40      AND #$40     Rechteck ausgewählt ?
7141   F0 0B      BEQ $714E    Nein, Skip
7143   8A         TXA
7144   AE 3A 12   LDX $123A
7147   9D 67 12   STA $1267,X  Impulsbreite setzen
714A   98         TYA
714B   9D 5D 12   STA $125D,X
714E   AE 3A 12   LDX $123A
7151   AD 3B 12   LDA $123B    Klangwerte wieder programmieren
7154   9D 3F 12   STA $123F,X
7157   AD 3C 12   LDA $123C
715A   9D 49 12   STA $1249,X
715D   AD 3D 12   LDA $123D
```

```
7160   9D 53 12   STA $1253,X
7163   60         RTS

******************************* BASIC-Befehl COLLISION

7164   20 F4 87   JSR $87F4    Typ auswerten
7167   CA         DEX
7168   E0 03      CPX #$03     Wert korrekt ?
716A   B0 21      BCS $718D    Nein, Fehler
716C   8E 80 12   STX $1280    Kollisionstyp setzen
716F   20 06 9E   JSR $9E06    Zeilennummer auswerten
7172   08         PHP          Status retten
7173   AE 80 12   LDX $1280
7176   9D 7C 12   STA $127C,X  Zeilennummer setzen
7179   98         TYA
717A   9D 79 12   STA $1279,X
717D   AD 7F 12   LDA $127F    IRQ-Flag laden
7180   1D B3 6C   ORA $6CB3,X  Kollisionsbit setzen
7183   28         PLP          Soll gesetzt werden ?
7184   B0 03      BCS $7189    Ja, Skip
7186   5D B3 6C   EOR $6CB3,X  Bit löschen
7189   8D 7F 12   STA $127F    und IRQ-Flaf speichern
718C   60         RTS
718D   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'

******************************* BASIC-Befehl SPRCOLOR

7190   C9 2C      CMP #$2C     ',' ?
7192   F0 0E      BEQ $71A2    Ja, Skip
7194   20 F4 87   JSR $87F4    Zusatzfarbe 1 holen
7197   CA         DEX
7198   E0 10      CPX #$10     Wert zu groß ?
719A   B0 17      BCS $71B3    Ja, Fehler
719C   20 45 A8   JSR $A845    ROMs einschalten
719F   8E 25 D0   STX $D025    und Zusatzfarbe setzen
71A2   20 1E 9E   JSR $9E1E    Zusatzfarbe 2 holen
71A5   90 0B      BCC $71B2    Wenn keine Angabe, Skip
71A7   CA         DEX
71A8   E0 10      CPX #$10     Wert zu groß ?
71AA   B0 07      BCS $71B3    Ja, Fehler
71AC   20 45 A8   JSR $A845    ROMs einschalten
71AF   8E 26 D0   STX $D026    Zusatzfarbe 2 setzen
71B2   60         RTS
71B3   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'

******************************* BASIC-Befehl WIDTH

71B6   20 F4 87   JSR $87F4    Byte-Wert holen
```

```
71B9   CA         DEX
71BA   E0 02      CPX #$02     Zu groß ?
71BC   B0 04      BCS $71C2    Ja, Fehler
71BE   8E 6B 11   STX $116B    Widthwert setzen
71C1   60         RTS
71C2   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'

****************************** BASIC-Befehl VOL

71C5   20 F4 87    SR $87F4    Byte-Wert holen
71C8   E0 10       PX #$10     Zu groß ?
71CA   B0 1D       CS $71E9    Ja, Fehler
71CC   86 77       TX $77      Wert setzen
71CE   AD 74 12    DA $1274    SID-Werte setzen
71D1   29 F0      AND #$F0     Filterwerte retten
71D3   05 77      ORA $77      Lautstärke setzen
71D5   8D 74 12   STA $1274
71D8   AD 75 12   LDA $1275
71DB   29 F0      AND #$F0     Filterwerte retten
71DD   05 77      ORA $77      Lautstärke setzen
71DF   8D 75 12   STA $1275
71E2   20 45 A8   JSR $A845    ROMs einschalten
71E5   8D 18 D4   STA $D418    und SID programmieren
71E8   60         RTS
71E9   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'

****************************** BASIC-Befehl SOUND

71EC   20 F4 87   JSR $87F4    Stimmennummer holen
71EF   CA         DEX
71F0   E0 03      CPX #$03     Wert falsch ?
71F2   90 03      BCC $71F7    Nein, Skip
71F4   4C 28 7D   JMP $7D28    'ILLEAGL QUANTITY'
71F7   8E 81 12   STX $1281    Stimme für Sound setzen
71FA   20 0F 88   JSR $880F    Frequenz auswerten
71FD   8C A5 12   STY $12A5    Maximalwert setzen
7200   8D A6 12   STA $12A6
7203   8C AC 12   STY $12AC    Frequenz setzen
7206   8D AD 12   STA $12AD
7209   20 0F 88   JSR $880F    Dauer auswerten
720C   C9 80      CMP #$80     Wert > 32767 ?
720E   B0 E4      BCS $71F4    Ja, Fehler
7210   8C A3 12   STY $12A3    Dauer setzen
7213   8D A4 12   STA $12A4
7216   20 1C 9E   JSR $9E1C    Richtung holen
7219   E0 03      CPX #$03     Wert zu groß ?
721B   B0 D7      BCS $71F4    Ja, Fehler
721D   8A         TXA
```

```
721E   8D A9 12   STA $12A9    Richtung setzen
7221   29 01      AND #$01     Bit 0 isolieren
7223   08         PHP          und Status retten
7224   20 06 9E   JSR $9E06    Maximalfrequenz holen
7227   8C A7 12   STY $12A7    und setzen
722A   8D A8 12   STA $12A8
722D   20 06 9E   JSR $9E06    Stufe holen
7230   28         PLP          Richtung abnehmend ?
7231   F0 0D      BEQ $7240    Nein, Skip
7233   48         PHA          Wert negieren
7234   98         TYA
7235   49 FF      EOR #$FF
7237   18         CLC
7238   69 01      ADC #$01
723A   A8         TAY
723B   68         PLA
723C   49 FF      EOR #$FF
723E   69 00      ADC #$00
7240   8D AB 12   STA $12AB    und Stufe setzen
7243   98         TYA
7244   8D AA 12   STA $12AA
7247   A2 02      LDX #$02     Welle mit Rechteck vorbesetzen
7249   20 1E 9E   JSR $9E1E    Wellenform holen
724C   E0 04      CPX #$04     Wert zu groß ?
724E   B0 CB      BCS $721B    Ja, Fehler
7250   BD B7 6C   LDA $6CB7,X  Bitmuster für Wellenform holen
7253   09 01      ORA #$01     KEY-Bit setzen
7255   8D B0 12   STA $12B0    und Wellenform speichern
7258   20 06 9E   JSR $9E06    Impulsbreite holen
725B   B0 04      BCS $7261    Wenn angegeben, Skip
725D   A9 08      LDA #$08     Impulsbreite = $0800 = 2048
725F   A0 00      LDY #$00
7261   C9 10      CMP #$10     Wert > 4095 ?
7263   B0 8F      BCS $71F4    Ja, Fehler
7265   8C AE 12   STY $12AE    Impulsbreite setzen
7268   8D AF 12   STA $12AF
726B   AD A3 12   LDA $12A3    Dauer = 0 ?
726E   0D A4 12   ORA $12A4
7271   F0 46      BEQ $72B9    Ja, Skip
7273   AE 81 12   LDX $1281    Stimmennummer
7276   8A         TXA          in Offset umrechnen
7277   0A         ASL
7278   A8         TAY
7279   B9 24 12   LDA $1224,Y  Stimme fertig ?
727C   10 FB      BPL $7279    Nein, warten
727E   BD 85 12   LDA $1285,X  Sound fertig ?
7281   10 FB      BPL $727E    Nein, warten
7283   A0 00      LDY #$00
```

```
7285   B9 A5 12   LDA $12A5,Y  Sounddaten programmieren
7288   9D 88 12   STA $1288,X
728B   E8         INX
728C   E8         INX
728D   E8         INX
728E   C8         INY
728F   C0 09      CPY #$09
7291   D0 F2      BNE $7285
7293   AE 81 12   LDX $1281    Stimmennummer
7296   BC 39 70   LDY $7039,X  Stimmenoffset für SID laden
7299   20 45 A8   JSR $A845    ROMs einschalten
729C   A9 08      LDA #$08     Stimme deblockieren (TEST-Bit setzen)
729E   99 04 D4   STA $D404,Y
72A1   A9 00      LDA #$00     Attack/Decay auf 0
72A3   99 05 D4   STA $D405,Y
72A6   A9 F0      LDA #$F0     Sustain auf Maximalstärke, Release auf 0
72A8   99 06 D4   STA $D406,Y
72AB   A2 00      LDX #$00
72AD   BD AC 12   LDA $12AC,X  SID mit Sounddaten programmieren
72B0   99 00 D4   STA $D400,Y
72B3   C8         INY
72B4   E8         INX
72B5   E0 05      CPX #$05
72B7   D0 F4      BNE $72AD
72B9   AE 81 12   LDX $1281    Stimmennummer laden
72BC   AC A3 12   LDY $12A3    Dauer laden
72BF   AD A4 12   LDA $12A4
72C2   78         SEI
72C3   9D 85 12   STA $1285,X  und Dauer programmieren
72C6   98         TYA
72C7   9D 82 12   STA $1282,X
72CA   58         CLI
72CB   60         RTS

******************************** BASIC-Befehl WINDOW

72CC   20 F4 87   JSR $87F4    Spalte links oben holen
72CF   E0 28      CPX #$28     Wert > 39 ?
72D1   24 D7      BIT $D7      80-Zeichen-Modus ?
72D3   10 02      BPL $72D7    Nein, Skip
72D5   E0 50      CPX #$50     Wert > 79 ?
72D7   B0 59      BCS $7332    Ja, Fehler
72D9   8E B3 12   STX $12B3    Spalte links oben setzen
72DC   20 09 88   JSR $8809    Zeile links oben holen
72DF   E0 19      CPX #$19     Wert zu groß ?
72E1   B0 4F      BCS $7332    Ja, Fehler
72E3   8E B4 12   STX $12B4    Zeile links oben setzen
72E6   20 09 88   JSR $8809    Spalte rechts unten holen
```

```
72E9   E0 28      CPX #$28     Wert > 39 ?
72EB   24 D7      BIT $D7      80-Zeichen-Modus ?
72ED   10 02      BPL $72F1    Nein, Skip
72EF   E0 50      CPX #$50     Wert > 79 ?
72F1   B0 3F      BCS $7332    Ja, Fehler
72F3   8E B5 12   STX $12B5    Spalte rechts unten setzen
72F6   EC B3 12   CPX $12B3    < Spalte links oben ?
72F9   90 37      BCC $7332    Ja, Fehler
72FB   20 09 88   JSR $8809    Zeile rechts unten holen
72FE   E0 19      CPX #$19     Wert zu groß ?
7300   B0 30      BCS $7332    Ja, Fehler
7302   8E B6 12   STX $12B6    Zeile rechts unten setzen
7305   EC B4 12   CPX $12B4    < Zeile links oben ?
7308   90 28      BCC $7332    Ja, Fehler
730A   20 1C 9E   JSR $9E1C    Löschwert holen
730D   E0 02      CPX #$02     Wert zu groß ?
730F   B0 21      BCS $7332    Ja, Fehler
7311   8A         TXA          Wert speichern
7312   48         PHA
7313   AE B3 12   LDX $12B3    Daten links oben laden
7316   AD B4 12   LDA $12B4
7319   18         CLC          Flag für links oben
731A   20 2D C0   JSR $C02D    Windowecke setzen
731D   AE B5 12   LDX $12B5    Daten für rechts unten laden
7320   AD B6 12   LDA $12B6
7323   38         SEC          Flag für rechts unten
7324   20 2D C0   JSR $C02D    Windowecke setzen
7327   A2 13      LDX #$13     Code für (HOME)
7329   68         PLA          Window löschen ?
732A   F0 02      BEQ $732E    Nein, Skip
732C   A2 93      LDX #$93     Code für (CLR)
732E   8A         TXA          Code in (A)
732F   4C 69 92   JMP $9269    und ausgeben
7332   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'

******************************* BASIC-Befehl BOOT

7335   A9 E6      LDA #$E6     Angaben für verbotene Parameter setzen
7337   A2 FC      LDX #$FC
7339   20 C3 A3   JSR $A3C3    und Diskparameter auswerten
733C   A5 80      LDA $80
733E   4A         LSR          Dateiname ?
733F   90 1D      BCC $735E    Nein, BOOT ohne Name
7341   20 1F A2   JSR $A21F    BLOAD-Befehl aufrufen
7344   B0 29      BCS $736F    Wenn Fehler, Skip
7346   AE D5 03   LDX $03D5    BASIC-Bankwert laden
7349   A5 81      LDA $81      Bank angegeben ?
734B   4A         LSR
```

```
734C   90 03     BCC $7351   Nein, Skip
734E   AE 1F 01  LDX $011F   Bank laden
7351   86 02     STX $02     Bank setzen
7353   A5 AC     LDA $AC     Startadresse setzen
7355   85 04     STA $04
7357   A5 AD     LDA $AD
7359   85 03     STA $03
735B   4C 6E FF  JMP $FF6E   JSRFAR Routine aufrufen
735E   AD 12 01  LDA $0112   Laufwerknummer laden
7361   09 30     ORA #$30    und in ASCII wandeln
7363   AE 1C 01  LDX $011C   Geräteadresse laden
7366   20 45 A8  JSR $A845   ROMs einschalten
7369   20 53 FF  JSR $FF53   BOOT-CALL
736C   B0 01     BCS $736F   Wenn Fehler, Skip
736E   60        RTS
736F   4C D0 90  JMP $90D0   Fehlerauswertung

******************************* BASIC-Befehl SPRDEF

7372   20 4F 9F  JSR $9F4F   HIRES-Bereich reservieren
7375   20 45 A8  JSR $A845   ROMs einschalten
7378   A9 D0     LDA #$D0    Zeiger auf Graphikzeichensatz
737A   8D 68 11  STA $1168   ab $D000 setzen
737D   A9 20     LDA #$20    Graphik einschalten
737F   85 D8     STA $D8
7381   20 30 6B  JSR $6B30   HIRES-Schirm löschen
7384   A0 80     LDY #$80
7386   8C 3D 11  STY $113D
7389   A0 18     LDY #$18    Y-Position = 24
738B   A9 20     LDA #$20    ' '
738D   A2 00     LDX #$00    X-Position = 0
738F   20 DB 68  JSR $68DB   (A) an (X)/(Y) in HIRES-Schirm
7392   E8        INX
7393   E0 15     CPX #$15    21 Spalten gelöscht ?
7395   90 F8     BCC $738F   Nein, Weiter
7397   20 DB 68  JSR $68DB   Zeichen in HIRES-Schirm
739A   88        DEY         24 Zeilen gelöscht ?
739B   10 FA     BPL $7397   Nein, weiter
739D   20 45 A8  JSR $A845   ROMs einschalten
73A0   A5 F1     LDA $F1     Farbcode unter Cursor retten
73A2   48        PHA
73A3   AD 21 D0  LDA $D021   Farbcode = Hintergrundfarbe
73A6   85 F1     STA $F1
73A8   A9 2B     LDA #$2B    '+'
73AA   A2 00     LDX #$00    X-Position = 0
73AC   8E 3D 11  STX $113D
73AF   A0 00     LDY #$00    Y-Position = 0
73B1   20 DB 68  JSR $68DB   Zeichen an (X)/(Y) in HIRES-Schirm
```

```
73B4   C8         INY
73B5   C0 18      CPY #$18      24 Spalten gesetzt ?
73B7   90 F8      BCC $73B1     Nein, weiter
73B9   E8         INX
73BA   E0 15      CPX #$15      21 Zeilen gefüllt ?
73BC   90 F1      BCC $73AF     Nein, weiter füllen
73BE   68         PLA
73BF   85 F1      STA $F1       Farbcode wieder setzen
73C1   20 D4 76   JSR $76D4
73C4   A0 02      LDY #$02      Zeile und Spalte anwählen
73C6   A2 17      LDX #$17
73C8   B9 6C 76   LDA $766C,Y   'SPRITE NUMMER?  '
73CB   F0 06      BEQ $73D3
73CD   20 DB 68   JSR $68DB     Zeichen an (X)/(Y) in Schirm
73D0   C8         INY           Y-Position erhöhen
73D1   D0 F5      BNE $73C8     Unbedingter Sprung
73D3   20 45 A8   JSR $A845     ROMs einschalten
73D6   20 E4 FF   JSR $FFE4     GETIN
73D9   F0 FB      BEQ $73D6     Wenn keine Eingabe, warten
73DB   C9 0D      CMP #$0D      (CR) ?
73DD   D0 08      BNE $73E7     Nein, Skip
73DF   20 30 6B   JSR $6B30     HIRES-Bereich löschen
73E2   A9 00      LDA #$00      Graphik ausschalten
73E4   85 D8      STA $D8
73E6   60         RTS
73E7   38         SEC
73E8   E9 31      SBC #$31
73EA   8D FC 12   STA $12FC     Spritenummer setzen
73ED   C9 08      CMP #$08      Wert falsch ?
73EF   B0 E5      BCS $73D6     Ja, dann weiter warten
73F1   AA         TAX
73F2   0A         ASL
73F3   A8         TAY
73F4   BD B3 6C   LDA $6CB3,X   Bitmuster für Sprite laden
73F7   8D 6D 11   STA $116D     und setzen
73FA   2D 1C D0   AND $D01C     Multicolormodus ?
73FD   F0 02      BEQ $7401     Nein, Skip
73FF   A9 80      LDA #$80      Multicolorflag laden
7401   8D FA 12   STA $12FA     und setzen
7404   A9 08      LDA #$08      X- und Y-Koordinate setzen
7406   99 D6 11   STA $11D6,Y
7409   A9 4A      LDA #$4A
740B   99 D7 11   STA $11D7,Y
740E   AD 6D 11   LDA $116D
7411   0D E6 11   ORA $11E6
7414   8D E6 11   STA $11E6     MSB der X-Koordinate setzen
7417   AD 6D 11   LDA $116D     Sprite einschalten
741A   8D 15 D0   STA $D015
```

```
741D   AE FC 12   LDX $12FC     Spritenummer laden
7420   BC D9 6D   LDY $6DD9,X   Offset auf Spritedaten laden
7423   A9 00      LDA #$00
7425   99 7E 11   STA $117E,Y   Spritegeschwindigkeit auf 0
7428   8A         TXA           Nummer in (A)
7429   A0 11      LDY #$11      Schirmposition 17/23
742B   A2 17      LDX #$17
742D   18         CLC           Nummer in ASCII wandeln
742E   69 31      ADC #$31
7430   20 DB 68   JSR $68DB     Spritenummer ausgeben
7433   20 45 A8   JSR $A845     ROMs einschalten
7436   AD FC 12   LDA $12FC
7439   4A         LSR
743A   6A         ROR
743B   6A         ROR
743C   85 4B      STA $4B       Aktuelle Spriteadresse setzen
743E   A0 0E      LDY #$0E
7440   90 01      BCC $7443
7442   C8         INY
7443   84 4C      STY $4C
7445   20 D1 75   JSR $75D1     Sprite anzeigen
7448   A0 3F      LDY #$3F
744A   B1 4B      LDA ($4B),Y   Aktuelles Sprite in Arbeitsspeicher
744C   99 B7 12   STA $12B7,Y
744F   88         DEY
7450   10 F8      BPL $744A
7452   A2 00      LDX #$00      SPRDEF-Cursor auf Homeposition
7454   8E 5F 11   STX $115F
7457   8E 5E 11   STX $115E
745A   20 4A 76   JSR $764A     Cursor setzen
745D   20 E4 FF   JSR $FFE4     GETIN Zeichen holen
7460   F0 FB      BEQ $745D     Wenn keine Eingabe, warten
7462   48         PHA           Eingabe retten
7463   20 4A 76   JSR $764A     Cursor wieder löschen
7466   68         PLA
7467   A2 10      LDX #$10
7469   DD B4 76   CMP $76B4,X   Eingabe einer Farbe ?
746C   D0 0D      BNE $747B     Nein, weiter testen
746E   CA         DEX
746F   8A         TXA           Farbe in (A)
7470   AE FC 12   LDX $12FC     Spritenummer laden
7473   9D 27 D0   STA $D027,X   und Spritefarbe setzen
7476   20 D1 75   JSR $75D1     Sprite anzeigen
7479   B0 DF      BCS $745A     Unbedingter Sprung

747B   CA         DEX
747C   D0 EB      BNE $7469
747E   A2 11      LDX #$11
```

```
7480   DD 7F 76   CMP $767F,X   Eingabe eines Kommandos ?
7483   F0 05      BEQ $748A     Ja, Kommando aufrufen
7485   CA         DEX
7486   10 F8      BPL $7480
7488   30 D0      BMI $745A     Unbedingter Sprung

748A   8A         TXA           Offset errechnen
748B   A8         TAY
748C   0A         ASL
748D   AA         TAX
748E   BD 91 76   LDA $7691,X   RTS-Adresse auf Stapel legen
7491   48         PHA
7492   BD 92 76   LDA $7692,X
7495   48         PHA
7496   60         RTS           und Kommando aufrufen

****************************** Punkt setzen (Befehle 1 bis 4)

7497   98         TYA           Befehlsnummer setzen
7498   85 8F      STA $8F
749A   20 10 76   JSR $7610     Spritefarbe holen
749D   48         PHA           und retten
749E   AC 5E 11   LDY $115E     Position laden
74A1   AE 5F 11   LDX $115F
74A4   20 C5 76   JSR $76C5     Schirmadresse setzen
74A7   68         PLA           Farbe holen
74A8   20 3F 76   JSR $763F     und setzen
74AB   AC 5E 11   LDY $115E
74AE   98         TYA
74AF   29 07      AND #$07
74B1   AA         TAX
74B2   98         TYA
74B3   4A         LSR
74B4   4A         LSR
74B5   4A         LSR
74B6   18         CLC
74B7   6D 5F 11   ADC $115F
74BA   6D 5F 11   ADC $115F
74BD   6D 5F 11   ADC $115F
74C0   A8         TAY
74C1   B1 4B      LDA ($4B),Y   Byte aus Sprite holen
74C3   2C FA 12   BIT $12FA     Multicolormodus ?
74C6   10 18      BPL $74E0     Nein, Skip
74C8   85 8E      STA $8E       Byte-Wert retten
74CA   BD 1C 9D   LDA $9D1C,X   Beide Multicolorbits setzen
74CD   1D 1D 9D   ORA $9D1D,X
74D0   48         PHA
74D1   05 8E      ORA $8E       und in Byte-Wert setzen
```

```
74D3   85 8E      STA $8E
74D5   68         PLA
74D6   A6 8F      LDX $8F
74D8   3D 25 9F   AND $9F25,X
74DB   45 8E      EOR $8E        Byte-Wert je nach Kommando ändern
74DD   4C EA 74   JMP $74EA      und wieder in Sprite setzen
74E0   1D 1C 9D   ORA $9D1C,X    Bit setzen
74E3   06 8F      ASL $8F        Kommando '1' ?
74E5   D0 03      BNE $74EA      Nein, Skip
74E7   5D 1C 9D   EOR $9D1C,X    Bit wieder löschen
74EA   91 4B      STA ($4B),Y    Byte-Wert wieder in Sprite
74EC   2C FA 12   BIT $12FA      Autorepeat ?
74EF   50 4E      BVC $753F      Ja, Skip
74F1   4C 5A 74   JMP $745A      Zum Schleifenanfang

****************************** Sprite speichern (SHIFT + CR)

74F4   A0 3F      LDY #$3F
74F6   B9 B7 12   LDA $12B7,Y    Sprite vom Arbeitsspeicher
74F9   91 4B      STA ($4B),Y    in Spritebereich übertragen
74FB   88         DEY
74FC   10 F8      BPL $74F6
74FE   A9 00      LDA #$00
7500   8D 15 D0   STA $D015      Sprite ausschalten
7503   4C??*a3    JMP $73C4      Zur Spritenummerneingabe

****************************** Spriteattributkommandos

7506   AD FA 12   LDA $12FA      Multicolorbit ändern
7509   49 80      EOR #$80
750B   8D FA 12   STA $12FA
750E   20 D1 75   JSR $75D1
7511   AD 5E 11   LDA $115E      und Position korrigieren
7514   29 FE      AND #$FE
7516   8D 5E 11   STA $115E
7519   A0 1C      LDY #$1C       Offset auf Multicolorregister
751B   2C         .BYTE $2C
751C   A0 17      LDY #$17       Offset auf Expand X
751E   2C         .BYTE $2C
751F   A0 1D      LDY #$1D       Offset auf Expand Y
7521   B9 00 D0   LDA $D000,Y    Aktuellen Wert holen
7524   AE FC 12   LDX $12FC      Spritenummer als Offset
7527   5D B3 6C   EOR $6CB3,X    Wert mit Spritebitmuster verknüpfen
752A   99 00 D0   STA $D000,Y    und wieder setzen
752D   4C 5A 74   JMP $745A      Zum Schleifenstart
```

```
******************************** Aktuelles Sprite löschen (CLR-Kommando)

7530   A0 3F      LDY #$3F
7532   A9 00      LDA #$00
7534   91 4B      STA ($4B),Y
7536   88         DEY
7537   10 FB      BPL $7534
7539   20 D4 76   JSR $76D4     Spriteschirm löschen
753C   4C 52 74   JMP $7452     Zum Schleifenstart

******************************** Cursorbewegungen für SPRDEF

753F   A9 01      LDA #$01      Cursor Right
7541   2C         .BYTE $2C

7542   A9 FF      LDA #$FF      Cursor Left
7544   2C FA 12   BIT $12FA     Multicolormodus ?
7547   10 01      BPL $754A     Nein, Skip
7549   0A         ASL           Wert * 2
754A   18         CLC
754B   6D 5E 11   ADC $115E     Wert auf Position aufaddieren
754E   30 09      BMI $7559     Wenn kleiner 0, Skip
7550   C9 18      CMP #$18      Position > 23 ?
7552   B0 2D      BCS $7581     Ja, Skip
7554   8D 5E 11   STA $115E     Neue Position setzen
7557   90 25      BCC $757E     Unbedingter Sprung
7559   A2 17      LDX #$17      Position auf 23
755B   2C FA 12   BIT $12FA     Multicolormodus ?
755E   10 01      BPL $7561     Nein, Skip
7560   CA         DEX           Position korrigieren
7561   8E 5E 11   STX $115E     neue Position setzen

7564   A9 FF      LDA #$FF      Cursor Up
7566   2C         .BYTE $2C

7567   A9 01      LDA #$01      Cursor Down
7569   18         CLC
756A   6D 5F 11   ADC $115F     Wert auf Position aufaddieren
756D   C9 15      CMP #$15      Wert > 20 ?
756F   B0 0D      BCS $757E     Ja, Skip
7571   8D 5F 11   STA $115F     neue Position setzen
7574   90 08      BCC $757E

******************************** Autorepeat (A-Kommando)

7576   AD FA 12   LDA $12FA     Autorepeatbit ändern
7579   49 40      EOR #$40
757B   8D FA 12   STA $12FA
```

```
757E   4C 5A 74   JMP $745A     Zum Schleifenstart

7581   A9 00      LDA #$00      Position auf 0
7583   8D 5E 11   STA $115E
7586   F0 DF      BEQ $7567     Unbedingter Sprung

****************************** Sprite kopieren

7588   A0 02      LDY #$02      Zeile und Spalte setzen
758A   A2 18      LDX #$18
758C   B9 61 76   LDA $7661,Y   'COPY FROM?'
758F   F0 06      BEQ $7597
7591   20 DB 68   JSR $68DB     in HIRES-Schirm
7594   C8         INY
7595   D0 F5      BNE $758C
7597   20 45 A8   JSR $A845     ROMs einschalten
759A   20 E4 FF   JSR $FFE4     GETIN Zeichen abwarten
759D   F0 FB      BEQ $759A     Wenn keine Eingabe, warten
759F   C9 0D      CMP #$0D      (CR) ?
75A1   F0 1F      BEQ $75C2     ja, Skip
75A3   38         SEC
75A4   E9 31      SBC #$31
75A6   C9 08      CMP #$08      Gültige Spritenummer ?
75A8   B0 F0      BCS $759A     Nein, weiter warten
75AA   4A         LSR
75AB   6A         ROR
75AC   6A         ROR
75AD   85 8E      STA $8E       Spriteadresse setzen
75AF   A0 0E      LDY #$0E
75B1   90 01      BCC $75B4
75B3   C8         INY
75B4   84 8F      STY $8F
75B6   A0 3F      LDY #$3F
75B8   B1 8E      LDA ($8E),Y   Sprite kopieren
75BA   91 4B      STA ($4B),Y
75BC   88         DEY
75BD   10 F9      BPL $75B8
75BF   20 D1 75   JSR $75D1
75C2   A9 00      LDA #$00      COPY-Meldung vom Anfang wieder löschen
75C4   A8         TAY
75C5   99 00 3E   STA $3E00,Y
75C8   88         DEY
75C9   D0 FA      BNE $75C5
75CB   4C 5A 74   JMP $745A
```

```
******************************** Füllbytes

75CE   FF FF FF

******************************** Sprite anzeigen

75D1   A2 00      LDX #$00
75D3   8E 60 11   STX $1160      Spaltenzähler löschen
75D6   8E FB 12   STX $12FB      Zeilenzähler löschen
75D9   20 C5 76   JSR $76C5      Schirmadresse setzen
75DC   A0 00      LDY #$00       Offset auf 0
75DE   A2 08      LDX #$08       Bitzähler setzen
75E0   8C 6E 11   STY $116E      (Y) retten
75E3   AC 60 11   LDY $1160      Spaltenoffset laden
75E6   B1 4B      LDA ($4B),Y    Byte aus Sprite laden
75E8   EE 60 11   INC $1160      Spaltenzähler erhöhen
75EB   AC 6E 11   LDY $116E      (Y) wieder holen
75EE   0A         ASL
75EF   2C FA 12   BIT $12FA      Multicolormodus ?
75F2   10 02      BPL $75F6      Nein, Skip
75F4   2A         ROL            Zweites Bit ebenfalls setzen
75F5   CA         DEX            Bitzähler anpassen
75F6   48         PHA
75F7   2A         ROL
75F8   20 10 76   JSR $7610      Spritefarbe holen
75FB   20 3F 76   JSR $763F      und Farbwerte in Schirm
75FE   C8         INY
75FF   68         PLA
7600   CA         DEX            Alle Bits angezeigt ?
7601   D0 EB      BNE $75EE      Nein, weitermachen
7603   C0 18      CPY #$18       Alle Spalten angezeigt ?
7605   90 D7      BCC $75DE      Nein, weitermachen
7607   AE FB 12   LDX $12FB      Alle Zeilen angezeigt ?
760A   E8         INX
760B   E0 15      CPX #$15
760D   90 C7      BCC $75D6      Nein weitermachen
760F   60         RTS

******************************** Spritefarbe (A) holen

7610   29 03      AND #$03       Farbnummer isolieren
7612   4A         LSR
7613   6A         ROR
7614   F0 0F      BEQ $7625      Wenn Hintergrundfarbe, Skip
7616   2C FA 12   BIT $12FA      Multicolormodus ?
7619   10 0F      BPL $762A      nein, Skip
761B   AD 25 D0   LDA $D025      Multicolorfarbe 1 laden
761E   90 14      BCC $7634      Wenn gewünscht, Skip
```

```
7620   AD 26 D0   LDA $D026     Multicolorfarbe 2 laden
7623   B0 0F      BCS $7634     Unbedingter Sprung
7625   AD 21 D0   LDA $D021     Hintergrundfarbe laden
7628   90 0A      BCC $7634     Unbedingter Sprung
762A   86 8E      STX $8E       (X) retten
762C   AE FC 12   LDX $12FC     Spritenummer laden
762F   BD 27 D0   LDA $D027,X   und Spritefarbe laden
7632   A6 8E      LDX $8E       (X) wieder holen
7634   29 0F      AND #$0F
7636   85 8E      STA $8E       Farbwert berechnen
7638   0A         ASL
7639   0A         ASL
763A   0A         ASL
763B   0A         ASL
763C   05 8E      ORA $8E
763E   60         RTS

******************************** Farbwert an SPRDEF-Cursorposition

763F   91 8C      STA ($8C),Y   Farbwert setzen
7641   2C FA 12   BIT $12FA     Multicolormodus ?
7644   10 03      BPL $7649     Nein, Ende
7646   C8         INY
7647   91 8C      STA ($8C),Y   Farbwert für zweites Zeichen setzen
7649   60         RTS

******************************** SPRDEF-Cursor an/aus

764A   AE 5F 11   LDX $115F     Zeile laden
764D   20 C5 76   JSR $76C5     Schirmadresse setzen
7650   AC 5E 11   LDY $115E     Spalte laden
7653   2C FA 12   BIT $12FA     Multicolormodus ?
7656   10 03      BPL $765B     Nein, Skip
7658   20 5B 76   JSR $765B     Aufruf für 2 Zeichen invertieren
765B   B1 8C      LDA ($8C),Y   Farbwert invertieren
765D   49 80      EOR #$80
765F   91 8C      STA ($8C),Y
7661   C8         INY
7662   60         RTS

******************************** Texte für SPRDEF

7663   43 4F 50 59 20 46 52 4F 'COPY FROM?'
766B   4D 3F 00

766E   53 50 52 49 54 45 20 4E 'SPRITE NUMBER?  '
7676   55 4D 42 45 52 3F 20 20
```

```
767E   00

******************************** Kommandotabelle für SPRDEF

767F   31 32 33 34 03 8D 58 59 '1234' (STOP) (SHIFT-CR) 'XY'
7687   4D 9D 1D 91 11 93 13 41 'M' (LEFT) (RIGHT) (CLR) (HOME) 'A'
768F   0D 43                   (CR) 'C'

******************************** RTS-Adressen für Kommandos

7691   74 96 74 96 74 96 74 96
7699   74 F3 74 FD 75 1E 75 1B
76A1   75 05 75 41 75 3E 75 63
76A9   75 66 75 2F 74 51 75 75
76B1   75 80 75 87

******************************** Tabelle für Farbkommandos

76B5   90 05 1C 9F 9C 1E 1F 9E
76BD   81 95 96 97 98 99 9A 9B

******************************** Textschirmadresse für Zeile (X) setzen

76C5   BD 33 C0   LDA $C033,X  Normale Textschirmadresse setzen
76C8   85 8C      STA $8C
76CA   BD 4C C0   LDA $C04C,X
76CD   29 03      AND #$03
76CF   09 1C      ORA #$1C     High-Byte auf $1C00-$1FFF setzen
76D1   85 8D      STA $8D
76D3   60         RTS

******************************** Spriteschirm mit Hintergrundfarbe füllen

76D4   AD 21 D0   LDA $D021    Hintergrundfarbe
76D7   20 34 76   JSR $7634    in Farbwert umrechnen
76DA   A2 14      LDX #$14     21 Zeilen löschen
76DC   48         PHA          Farbwert retten
76DD   20 C5 76   JSR $76C5    Textschirmadresse setzen
76E0   68         PLA          Farbwert holen
76E1   A0 17      LDY #$17     23 Spalten löschen
76E3   91 8C      STA ($8C),Y  Farbwert setzen
76E5   88         DEY          Alle Spalten gelöscht ?
76E6   10 FB      BPL $76E3    Nein, weiter
76E8   CA         DEX          Alle Zeilen gelöscht ?
76E9   10 F1      BPL $76DC    Nein, nächste Zeile
76EB   60         RTS
```

```
******************************** BASIC-Befehl SPRSAV

76EC    20 7C 77    JSR $777C      Ersten Parameter holen
76EF    B0 2F       BCS $7720      Wenn String, Skip
76F1    85 4B       STA $4B        Spriteadresse setzen
76F3    84 4C       STY $4C
76F5    A0 3E       LDY #$3E
76F7    B1 4B       LDA ($4B),Y    Spritedaten in Arbeitsspeicher
76F9    99 B7 12    STA $12B7,Y
76FC    88          DEY
76FD    10 F8       BPL $76F7
76FF    C8          INY
7700    8C F7 12    STY $12F7      Shapegröße High-Bytes auf 0 setzen
7703    8C F9 12    STY $12F9
7706    A9 17       LDA #$17       Low-Byte X auf 23
7708    8D F6 12    STA $12F6
770B    A9 14       LDA #$14       Low-Byte Y auf 20
770D    8D F8 12    STA $12F8
7710    A2 B7       LDX #$B7       Spritedatenadresse setzen
7712    A0 12       LDY #$12
7714    86 70       STX $70
7716    84 71       STY $71
7718    A9 43       LDA #$43       Stringlänge laden
771A    20 CC 86    JSR $86CC      Platz für String reservieren
771D    20 99 77    JSR $7799      Stringparameter holen

7720    8E DB 03    STX $03DB      Länge setzen
7723    8D DC 03    STA $03DC      und Adresse setzen
7726    8C DD 03    STY $03DD
7729    20 5C 79    JSR $795C      Test auf ','
772C    A5 3D       LDA $3D        PC retten
772E    8D E0 03    STA $03E0
7731    A5 3E       LDA $3E
7733    8D E1 03    STA $03E1
7736    20 7C 77    JSR $777C      Zweiten Parameter holen
7739    B0 25       BCS $7760      Wenn String, Skip
773B    85 8C       STA $8C        Spriteadresse setzen
773D    84 8D       STY $8D
773F    AD DC 03    LDA $03DC      Stringadresse setzen
7742    85 4B       STA $4B
7744    AD DD 03    LDA $03DD
7747    85 4C       STA $4C
7749    A0 00       LDY #$00
774B    CC DB 03    CPY $03DB      Länge erreicht ?
774E    F0 0F       BEQ $775F      Ja, Ende
7750    A9 4B       LDA #$4B
7752    20 AB 03    JSR $03AB
7755    8D 03 FF    STA $FF03
```

```
7758   91 8C      STA ($8C),Y  String übertragen
775A   C8         INY
775B   C0 3F      CPY #$3F
775D   D0 EC      BNE $774B
775F   60         RTS

7760   AD E0 03   LDA $03E0    PC wieder holen
7763   85 3D      STA $3D
7765   AD E1 03   LDA $03E1
7768   85 3E      STA $3E
776A   20 AF 7A   JSR $7AAF    Variable suchen/anlegen
776D   85 4B      STA $4B      Adresse setzen
776F   84 4C      STY $4C
7771   A9 DB      LDA #$DB     Deskriptoradresse setzen
7773   85 66      STA $66
7775   A9 03      LDA #$03
7777   85 67      STA $67
7779   4C 05 54   JMP $5405    String Wertzuweisung

******************************* Parameter für SPRSAV auswerten

777C   20 EF 77   JSR $77EF    FRMEVL Ausdruck auswerten
777F   24 0F      BIT $0F      Stringausdruck ?
7781   30 16      BMI $7799    Ja, Skip
7783   20 F7 87   JSR $87F7    Spritenummer holen
7786   CA         DEX
7787   E0 08      CPX #$08     Wert zu groß ?
7789   B0 0B      BCS $7796    Ja, Fehler
778B   8A         TXA
778C   4A         LSR          Spriteadresse in (A)/(Y)
778D   6A         ROR
778E   6A         ROR
778F   A0 0E      LDY #$0E
7791   90 01      BCC $7794
7793   C8         INY
7794   18         CLC          Flag für Spritenummer geholt
7795   60         RTS
7796   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'

******************************* Aktuelle Stringparameter holen

7799   A5 66      LDA $66
779B   A4 67      LDY $67
779D   20 E0 87   JSR $87E0    Stringzeiger holen
77A0   A0 00      LDY #$00
77A2   20 E7 42   JSR $42E7    Länge holen
77A5   AA         TAX          und in (X)
77A6   C8         INY
```

```
77A7   20 E7 42   JSR $42E7    Adresse in (A)/(Y) holen
77AA   48         PHA
77AB   C8         INY
77AC   20 E7 42   JSR $42E7
77AF   A8         TAY
77B0   68         PLA
77B1   38         SEC          Flag für String geholt
77B2   60         RTS

******************************* BASIC-Befehl FAST

77B3   20 45 A8   JSR $A845    ROMs einschalten
77B6   AD 11 D0   LDA $D011    VIC-Schirm ausschalten
77B9   29 6F      AND #$6F
77BB   8D 11 D0   STA $D011
77BE   A9 01      LDA #$01     2 Megahertz einschalten
77C0   8D 30 D0   STA $D030
77C3   60         RTS

******************************* BASIC-Befehl SLOW

77C4   20 45 A8   JSR $A845    ROMs einschalten
77C7   A9 00      LDA #$00     1 Megahertz einschalten
77C9   8D 30 D0   STA $D030
77CC   AD 11 D0   LDA $D011    VIC-Schirm wieder anschalten
77CF   29 7F      AND #$7F
77D1   09 10      ORA #$10
77D3   8D 11 D0   STA $D011
77D6   60         RTS

******************************* FRMNUM Numerischen Ausdruck auswerten

77D7   20 EF 77   JSR $77EF    FRMEVL Ausdruck auswerten

******************************* Auf numerisch testen

77DA   18         CLC          Flag für numerisch
77DB   90 01      BCC $77DE    Unbedingter Sprung

******************************* Auf String prüfen

77DD   38         SEC          Flag für String
77DE   24 0F      BIT $0F      Typ = String ?
77E0   30 03      BMI $77E5    Ja, Skip
77E2   B0 03      BCS $77E7    Wenn Stringtest, Fehler
77E4   60         RTS
77E5   B0 FD      BCS $77E4    Wenn Stringtest, Ende
77E7   A2 16      LDX #$16     'TYPE MISMATCH'
```

```
77E9   2C         .BYTE $2C
77EA   A2 19      LDX #$19      'FORMULA TOO COMPLEX'
77EC   4C 3C 4D   JMP $4D3C

****************************** FRMEVL Ausdruck auswerten

77EF   A6 3D      LDX $3D       PC erniedrigen
77F1   D0 02      BNE $77F5
77F3   C6 3E      DEC $3E
77F5   C6 3D      DEC $3D
77F7   A2 00      LDX #$00
77F9   24         .BYTE $24
77FA   48         PHA           Vergleichsflag auf Stapel legen
77FB   8A         TXA
77FC   48         PHA           Hierarchiewert setzen
77FD   BA         TSX           Stack voll ?
77FE   E0 63      CPX #$63
7800   90 E8      BCC $77EA     Ja, Fehler
7802   20 D7 78   JSR $78D7     Nächstes Element auswerten
7805   A9 00      LDA #$00      Vergleichsmaske löschen
7807   85 4F      STA $4F
7809   20 86 03   JSR $0386     CHRGOT
780C   38         SEC
780D   E9 B1      SBC #$B1      < Token für '>' ?
780F   90 17      BCC $7828     Ja, Skip
7811   C9 03      CMP #$03      > Token für '<' ?
7813   B0 13      BCS $7828     Ja, kein Vergleich, Skip
7815   C9 01      CMP #$01      Wenn '<','=', SEC
7817   2A         ROL           Carry als Bit 0 setzen
7818   49 01      EOR #$01      und Bit 0 invertieren
781A   45 4F      EOR $4F       Entsprechendes Bit in Maske
781C   C5 4F      CMP $4F       schon gesetzt ?
781E   90 61      BCC $7881     Ja, Fehler
7820   85 4F      STA $4F       Vergleichsmaske setzen
7822   20 80 03   JSR $0380     CHRGET
7825   4C 0C 78   JMP $780C     und weiterverarbeiten

7828   A6 4F      LDX $4F       Vergleich ?
782A   D0 2C      BNE $7858     Ja, Skip
782C   B0 7E      BCS $78AC     Wenn unpassendes Zeichen, Endeflag setzen
782E   69 07      ADC #$07      Rechenzeichen ?
7830   90 7A      BCC $78AC     Nein, Endeflag setzen
7832   65 0F      ADC $0F       Numerische Werte ?
7834   D0 03      BNE $7839     Ja, Skip
7836   4C 0D 87   JMP $870D     Stringverknüpfung
7839   69 FF      ADC #$FF      Wert in Offset umrechnen
783B   85 24      STA $24
783D   0A         ASL
```

```
783E   65 24      ADC $24
7840   A8         TAY          Offset = Wert * 3
7841   68         PLA          Letzten Hierarchiewert holen
7842   D9 28 48   CMP $4828,Y  < Hierarchieflag aus Tabelle ?
7845   B0 6A      BCS $78B1    Nein, Werte vom Stack holen
7847   20 DA 77   JSR $77DA    Test auf numerisch
784A   48         PHA          Hierarchiewert retten
784B   20 71 78   JSR $7871    Rekursiver Aufruf nach Setzen der
                               Operandenadresse
784E   68         PLA          Hierarchiewert wieder holen
784F   A4 4D      LDY $4D      Endeflag gesetzt ?
7851   10 17      BPL $786A    Nein, Skip
7853   AA         TAX          Hierarchiewert testen
7854   F0 59      BEQ $78AF    Wenn '+', Skip
7856   D0 62      BNE $78BA    Sonst Werte vom Stack holen

7858   46 0F      LSR $0F      Stringflag löschen
785A   8A         TXA          Vergleichsmaske
785B   2A         ROL          verdoppeln
785C   A6 3D      LDX $3D      PC erniedrigen
785E   D0 02      BNE $7862
7860   C6 3E      DEC $3E
7862   C6 3D      DEC $3D
7864   A0 1B      LDY #$1B     Offset auf Hierarchietabelle
7866   85 4F      STA $4F      Flag setzen
7868   D0 D7      BNE $7841    Unbedingter Sprung

786A   D9 28 48   CMP $4828,Y  > Hierarchiewert aus Tabelle ?
786D   B0 4B      BCS $78BA    Ja, Werte vom Stack holen
786F   90 D9      BCC $784A

7871   B9 2A 48   LDA $482A,Y  Operandenadresse auf Stapel legen
7874   48         PHA
7875   B9 29 48   LDA $4829,Y
7878   48         PHA
7879   20 84 78   JSR $7884    Operanden auf Stapel legen
787C   A5 4F      LDA $4F      Vergleichsflag laden
787E   4C FA 77   JMP $77FA    und zum Schleifenstart

7881   4C 6C 79   JMP $796C    'SYNTAX'

7884   A5 68      LDA $68      Vorzeichen von FAC#1 laden
7886   BE 28 48   LDX $4828,Y  Hierarchieflag laden
7889   A8         TAY
788A   18         CLC
788B   68         PLA          Aufrufadresse merken
788C   69 01      ADC #$01
788E   85 24      STA $24
```

```
7890   68         PLA
7891   69 00      ADC #$00
7893   85 25      STA $25
7895   98         TYA
7896   48         PHA
7897   20 47 8C   JSR $8C47     FAC#1 runden
789A   A5 67      LDA $67       und auf Stapel legen
789C   48         PHA
789D   A5 66      LDA $66
789F   48         PHA
78A0   A5 65      LDA $65
78A2   48         PHA
78A3   A5 64      LDA $64
78A5   48         PHA
78A6   A5 63      LDA $63
78A8   48         PHA
78A9   6C 24 00   JMP ($0024)   Zum Aufruf zurück (simuliertes RTS)

78AC   A0 FF      LDY #$FF      Endewert
78AE   68         PLA           Hierarchieflag = 0 ?
78AF   F0 23      BEQ $78D4     Ja, entspricht leerem Stapel, Skip
78B1   C9 64      CMP #$64      Aufruf durch '+' ?
78B3   F0 03      BEQ $78B8     Ja, Skip
78B5   20 DA 77   JSR $77DA     Test auf numerisch
78B8   84 4D      STY $4D       Flag setzen
78BA   68         PLA           Vergleichsflag holen
78BB   4A         LSR           wieder korrigieren
78BC   85 14      STA $14       und setzen
78BE   68         PLA           FAC#2 vom Stapel holen
78BF   85 6A      STA $6A
78C1   68         PLA
78C2   85 6B      STA $6B
78C4   68         PLA
78C5   85 6C      STA $6C
78C7   68         PLA
78C8   85 6D      STA $6D
78CA   68         PLA
78CB   85 6E      STA $6E
78CD   68         PLA
78CE   85 6F      STA $6F
78D0   45 68      EOR $68       Vorzeichen vergleichen
78D2   85 70      STA $70
78D4   A5 63      LDA $63       Exponent FAC#1 laden
78D6   60         RTS

****************************** Nächstes Element eines Ausdrucks holen

78D7   6C 0A 03   JMP ($030A)   $78DA Vektor Ausdruck auswerten
```

```
78DA   A9 00      LDA #$00     Typflag auf numerisch setzen
78DC   85 0F      STA $0F
78DE   20 80 03   JSR $0380    CHRGET
78E1   B0 05      BCS $78E8    Wenn keine Zahl, Skip
78E3   A2 00      LDX #$00     Flag für Bank 0
78E5   4C 22 8D   JMP $8D22    ASCII-Zahl in FAC#1 laden
78E8   20 3C 7B   JSR $7B3C    Buchstabe ?
78EB   90 03      BCC $78F0    Nein, Skip
78ED   4C 78 79   JMP $7978    Variable holen
78F0   C9 FF      CMP #$FF     Token für Pi ?
78F2   D0 0F      BNE $7903    Nein, Skip
78F4   A9 FE      LDA #$FE     Zeiger auf Konstante Pi
78F6   A0 78      LDY #$78
78F8   20 D4 8B   JSR $8BD4    FAC#1 = Konstante (A)/(Y)
78FB   4C 80 03   JMP $0380    Token für Pi überlesen

******************************* Konstante Pi

78FE   82 49 0F DA A1          3.14159265

******************************* Zeichen weiter testen

7903   C9 2E      CMP #$2E     '.' ?
7905   F0 DC      BEQ $78E3    Ja, Zahl auswerten
7907   C9 AB      CMP #$AB     '-' ?
7909   F0 66      BEQ $7971    Ja, Vorzeichenwechsel
790B   C9 AA      CMP #$AA     '+' ?
790D   F0 CF      BEQ $78DE    Ja, überlesen
790F   C9 22      CMP #$22     '"' ?
7911   D0 15      BNE $7928    Nein, Skip
7913   A5 3D      LDA $3D      PC auf erstes Zeichen des Strings setzen
7915   A4 3E      LDY $3E
7917   69 00      ADC #$00
7919   90 01      BCC $791C
791B   C8         INY
791C   20 9A 86   JSR $869A    String übernehmen
791F   A6 72      LDX $72      PC auf Stringende + 1 setzen
7921   A4 73      LDY $73
7923   86 3D      STX $3D
7925   84 3E      STY $3E
7927   60         RTS

7928   C9 A8      CMP #$A8     Token für NOT ?
792A   D0 16      BNE $7942    Nein, Skip
792C   A0 18      LDY #$18     Offset auf Hierarchietabelle laden
792E   D0 43      BNE $7973    Unbedingter Sprung
```

```
******************************** BASIC-Befehl NOT

7930   20 B4 84   JSR $84B4    FAC#1 in Integer wandeln
7933   A5 67      LDA $67      Wert invertieren
7935   49 FF      EOR #$FF
7937   A8         TAY
7938   A5 66      LDA $66
793A   49 FF      EOR #$FF
793C   20 E5 84   JSR $84E5    Flags für Umwandlung in FAC#1 setzen
793F   4C 70 8C   JMP $8C70    FAC#1 = Integer

********************************

7942   C9 A5      CMP #$A5     Token für FN ?
7944   D0 03      BNE $7949    Nein, Skip
7946   4C 3B 85   JMP $853B    Funktion übernehmen

7949   C9 B4      CMP #$B4     Funktion ?
794B   90 03      BCC $7950    Nein, Skip
794D   4C F7 4B   JMP $4BF7    Funktionen auswerten

******************************** Ausdruck in Klammern auswerten

7950   20 59 79   JSR $7959    Test auf '('
7953   20 EF 77   JSR $77EF    FRMEVL Ausdruck auswerten

******************************** Zeichen im Programmtext prüfen

7956   A9 29      LDA #$29     ')'
7958   2C         .BYTE $2C
7959   A9 28      LDA #$28     '('
795B   2C         .BYTE $2C
795C   A9 2C      LDA #$2C     ','
795E   A0 00      LDY #$00     Offset auf 0 setzen
7960   85 79      STA $79      Code setzen
7962   20 C9 03   JSR $03C9    Zeichen aus Programm holen
7965   C5 79      CMP $79      = Code ?
7967   D0 03      BNE $796C    Nein, Fehler
7969   4C 80 03   JMP $0380    Zeichen überlesen
796C   A2 0B      LDX #$0B     'SYNTAX'
796E   4C 3C 4D   JMP $4D3C

7971   A0 15      LDY #$15     Offset für Vorzeichenwechsel
7973   68         PLA          Rücksprungadresse löschen
7974   68         PLA
7975   4C 4B 78   JMP $784B    und in FRMEVL-Schleife
```

```
******************************* Variable holen

7978   20 AF 7A   JSR $7AAF   Variable suchen/anlegen
797B   85 66      STA $66     Deskriptoradresse setzen
797D   84 67      STY $67
797F   A6 47      LDX $47     Variablennamen laden
7981   A4 48      LDY $48
7983   A5 0F      LDA $0F     Variable numerisch ?
7985   F0 63      BEQ $79EA   Ja, Skip
7987   A9 00      LDA #$00    FAC#1 Rundungsstelle auf 0
7989   85 71      STA $71
798B   E0 54      CPX #$54    TI$ ?
798D   D0 25      BNE $79B4   Nein, Skip
798F   C0 C9      CPY #$C9
7991   D0 20      BNE $79B3   Nein, Skip
7993   A5 66      LDA $66     Zeiger auf Interpreterwert
7995   C9 D2      CMP #$D2    ab $03D2 ?
7997   D0 1A      BNE $79B3   Nein, Skip
7999   A5 67      LDA $67
799B   C9 03      CMP #$03    Nein, Skip
799D   D0 14      BNE $79B3
799F   20 1A 7A   JSR $7A1A   Zeit auswerten
79A2   84 60      STY $60     und setzen
79A4   88         DEY
79A5   84 72      STY $72
79A7   A0 06      LDY #$06    TI$ ist 6 Zeichen lang
79A9   84 5F      STY $5F
79AB   A0 24      LDY #$24    Offset auf Tabelle
79AD   20 CD 8E   JSR $8ECD   FAC#1 in ASCII wandeln
79B0   4C B8 85   JMP $85B8   und als String übernehmen
79B3   60         RTS

79B4   E0 44      CPX #$44    DS$ ?
79B6   D0 FB      BNE $79B3   Nein, Skip
79B8   C0 D3      CPY #$D3
79BA   D0 F7      BNE $79B3   Nein, Skip
79BC   20 E3 79   JSR $79E3   DS$ lesen falls nötig
79BF   A0 FF      LDY #$FF    Offset auf Start
79C1   C8         INY
79C2   A9 7B      LDA #$7B
79C4   20 AB 03   JSR $03AB   LDA ($7B),Y
79C7   C9 00      CMP #$00    Ende von DS$ erreicht ?
79C9   D0 F6      BNE $79C1   Nein, weiter suchen
79CB   98         TYA         Länge in (A)
79CC   20 88 86   JSR $8688   Stringzeiger berechnen
79CF   A8         TAY         Länge = 0 ?
79D0   F0 0E      BEQ $79E0   Ja, Skip
79D2   88         DEY         DS$ in Stringbereich übertragen
```

```
79D3   A9 7B      LDA #$7B
79D5   20 AB 03   JSR $03AB
79D8   91 37      STA ($37),Y
79DA   98         TYA
79DB   D0 F5      BNE $79D2
79DD   20 71 87   JSR $8771     Zeiger ($37) erhöhen
79E0   4C E3 86   JMP $86E3     und Zeiger in Deskriptorstack legen

****************************** DS$ lesen falls nötig

79E3   A5 7A      LDA $7A       DS$ schon gesetzt ?
79E5   D0 40      BNE $7A27     Ja, Ende
79E7   4C 78 A7   JMP $A778     DS$ einlesen

****************************** Numerische Variable holen

79EA   24 10      BIT $10       Integervariable ?
79EC   10 0F      BPL $79FD     Nein, Skip

79EE   A0 00      LDY #$00
79F0   20 E7 42   JSR $42E7     Wert in (A)/(Y) holen
79F3   AA         TAX
79F4   C8         INY
79F5   20 E7 42   JSR $42E7
79F8   A8         TAY
79F9   8A         TXA
79FA   4C 3C 79   JMP $793C     und in FAC#1

79FD   A5 67      LDA $67       Zeiger auf Interpreterwert
79FF   C9 03      CMP #$03      ab $03D2 ?
7A01   D0 7E      BNE $7A81     Nein, Skip
7A03   A5 66      LDA $66
7A05   C9 D2      CMP #$D2
7A07   D0 78      BNE $7A81     Nein, Skip
7A09   E0 54      CPX #$54      TI ?
7A0B   D0 1B      BNE $7A28     Nein, Skip
7A0D   C0 49      CPY #$49
7A0F   D0 70      BNE $7A81     Nein, Skip
7A11   20 1A 7A   JSR $7A1A     Zeit in FAC#1 holen
7A14   98         TYA           und in REAL wandeln
7A15   A2 A0      LDX #$A0
7A17   4C 7B 8C   JMP $8C7B

****************************** Zeit in FAC#1 holen

7A1A   20 DE FF   JSR $FFDE     RDTIM Zeit holen
7A1D   86 66      STX $66       und in FAC#1 setzen
7A1F   84 65      STY $65
```

```
7A21   85 67      STA $67
7A23   A0 00      LDY #$00
7A25   84 64      STY $64
7A27   60         RTS

*******************************

7A28   E0 53      CPX #$53    ST ?
7A2A   D0 0A      BNE $7A36   Nein, Skip
7A2C   C0 54      CPY #$54
7A2E   D0 51      BNE $7A81   Nein, Skip
7A30   20 B7 FF   JSR $FFB7   READST Status lesen
7A33   4C 68 8C   JMP $8C68   und in FAC#1

7A36   E0 44      CPX #$44    DS ?
7A38   D0 26      BNE $7A60   Nein, Skip
7A3A   C0 53      CPY #$53
7A3C   D0 43      BNE $7A81   Nein, Skip
7A3E   20 E3 79   JSR $79E3   DS$ lesen falls nötig
7A41   A0 00      LDY #$00    Offset auf Fehlernummer (1. Ziffer)
7A43   A9 7B      LDA #$7B
7A45   20 AB 03   JSR $03AB   LDA ($7B),Y
7A48   29 0F      AND #$0F    Numerischen Wert 0-9 isolieren
7A4A   0A         ASL         * 2
7A4B   85 11      STA $11
7A4D   0A         ASL         * 8
7A4E   0A         ASL
7A4F   65 11      ADC $11     * 10
7A51   85 11      STA $11     und Wert * 10 speichern
7A53   C8         INY
7A54   A9 7B      LDA #$7B
7A56   20 AB 03   JSR $03AB   2. Ziffer laden
7A59   29 0F      AND #$0F    Numerischen Wert isolieren
7A5B   65 11      ADC $11     Werte der Ziffern verknüpfen
7A5D   4C 68 8C   JMP $8C68   und in FAC#1 holen

7A60   E0 45      CPX #$45    ER oder EL ?
7A62   D0 1D      BNE $7A81   nein, Skip
7A64   C0 52      CPY #$52    ER ?
7A66   F0 10      BEQ $7A78   Ja, Skip
7A68   C0 4C      CPY #$4C    EL ?
7A6A   D0 15      BNE $7A81   Nein, Skip

******************************* EL in FAC#1 holen

7A6C   8D 03 FF   STA $FF03   Write in Bank 0 setzen
7A6F   AD 0A 12   LDA $120A   Wert EL holen
7A72   AC 09 12   LDY $1209
```

```
7A75   4C C9 84   JMP $84C9    und in FAC#1

****************************** ER in FAC#1 holen

7A78   8D 03 FF   STA $FF03    Write in Bank 0 setzen
7A7B   AD 08 12   LDA $1208    Wert von ER holen
7A7E   4C 68 8C   JMP $8C68    und in FAC#1

****************************** REAL-Variablenwert in FAC#1 holen

7A81   A5 66      LDA $66      Deskriptoradresse laden
7A83   A4 67      LDY $67

****************************** Konstante (A)/(Y) aus Bank 1 in FAC#1

7A85   85 24      STA $24      Adresse setzen
7A87   84 25      STY $25
7A89   A0 00      LDY #$00
7A8B   20 B7 03   JSR $03B7    Exponent übertragen
7A8E   85 63      STA $63
7A90   84 71      STY $71      Rundungsbyte löschen
7A92   C8         INY
7A93   20 B7 03   JSR $03B7    Vorzeichen übertragen
7A96   85 68      STA $68
7A98   09 80      ORA #$80
7A9A   85 64      STA $64
7A9C   C8         INY
7A9D   20 B7 03   JSR $03B7    Mantisse übertragen
7AA0   85 65      STA $65
7AA2   C8         INY
7AA3   20 B7 03   JSR $03B7
7AA6   85 66      STA $66
7AA8   C8         INY
7AA9   20 B7 03   JSR $03B7
7AAC   85 67      STA $67
7AAE   60         RTS

****************************** Variable suchen/anlegen

7AAF   A2 00      LDX #$00     Flag für nicht DIM-Befehl
7AB1   20 86 03   JSR $0386    CHRGOT
7AB4   86 0E      STX $0E      DIM-Flag setzen
7AB6   85 47      STA $47      Erstes Zeichen des Namens setzen
7AB8   20 86 03   JSR $0386    CHRGOT
7ABB   20 3C 7B   JSR $7B3C    Buchstabe ?
7ABE   B0 03      BCS $7AC3    Ja, Skip
7AC0   4C 6C 79   JMP $796C    'SYNTAX'
7AC3   A2 00      LDX #$00     Typflag
```

```
7AC5   86 0F      STX $0F     auf numerisch
7AC7   86 10      STX $10     und REAL
7AC9   20 80 03   JSR $0380   Nächstes Zeichen holen
7ACC   90 05      BCC $7AD3   Wenn Zahl, Skip
7ACE   20 3C 7B   JSR $7B3C   Buchstabe ?
7AD1   90 0B      BCC $7ADE   Nein, Skip
7AD3   AA         TAX         Zeichen retten
7AD4   20 80 03   JSR $0380   CHRGET
7AD7   90 FB      BCC $7AD4   Wenn Ziffer, überlesen
7AD9   20 3C 7B   JSR $7B3C   Buchstabe ?
7ADC   B0 F6      BCS $7AD4   Ja, überlesen
7ADE   C9 24      CMP #$24    '$' ?
7AE0   D0 06      BNE $7AE8   Nein, Skip
7AE2   A9 FF      LDA #$FF    Typflag auf String setzen
7AE4   85 0F      STA $0F
7AE6   D0 10      BNE $7AF8   Unbedingter Sprung
7AE8   C9 25      CMP #$25    '%' ?
7AEA   D0 13      BNE $7AFF   Nein, Skip
7AEC   A5 12      LDA $12     Integervariablen erlaubt ?
7AEE   D0 D0      BNE $7AC0   Nein, Fehler
7AF0   A9 80      LDA #$80    Integerflag setzen
7AF2   85 10      STA $10
7AF4   05 47      ORA $47     und Bit 7 im ersten Zeichen setzen
7AF6   85 47      STA $47
7AF8   8A         TXA         Zweites Zeichen holen
7AF9   09 80      ORA #$80    und Bit 7 für String setzen
7AFB   AA         TAX         Zeichen retten
7AFC   20 80 03   JSR $0380   CHRGOT
7AFF   86 48      STX $48     und zweites Zeichen setzen
7B01   38         SEC
7B02   05 12      ORA $12
7B04   E9 28      SBC #$28    Eingelesenes Zeichen = '(' ?
7B06   D0 03      BNE $7B0B   Nein, Skip
7B08   4C AB 7C   JMP $7CAB   Array lesen
7B0B   A0 00      LDY #$00
7B0D   84 12      STY $12     Integer erlauben
7B0F   A5 2F      LDA $2F     Zeiger auf Variablenstart
7B11   A6 30      LDX $30
7B13   86 62      STX $62     in Suchzeiger setzen
7B15   85 61      STA $61
7B17   E4 32      CPX $32     Ende Variablen erreicht ?
7B19   D0 04      BNE $7B1F   Nein, Skip
7B1B   C5 31      CMP $31
7B1D   F0 27      BEQ $7B46   Ja, Variable neu anlegen
7B1F   20 00 43   JSR $4300   Erstes Zeichen laden
7B22   C5 47      CMP $47     Variable gefunden ?
7B24   D0 0C      BNE $7B32   Nein, Skip
7B26   C8         INY
```

```
7B27   20 00 43   JSR $4300     Zweites Zeichen laden
7B2A   C5 48      CMP $48
7B2C   D0 03      BNE $7B31     Nein, Skip
7B2E   4C 57 7C   JMP $7C57
7B31   88         DEY
7B32   18         CLC           Zeiger auf nächste Variable setzen
7B33   A5 61      LDA $61
7B35   69 07      ADC #$07
7B37   90 DC      BCC $7B15
7B39   E8         INX
7B3A   D0 D7      BNE $7B13     und weitersuchen

****************************** Test auf Buchstabe ?

7B3C   C9 41      CMP #$41      < 'A' ?
7B3E   90 05      BCC $7B45     Ja, CLC Ende
7B40   E9 5B      SBC #$5B      Falls > 'Z', CLC Ende
7B42   38         SEC           sonst SEC
7B43   E9 A5      SBC #$A5
7B45   60         RTS

****************************** Variable neu anlegen

7B46   BA         TSX
7B47   BD 02 01   LDA $0102,X   Aufrufadresse
7B4A   C9 83      CMP #$83      von FRMEVL
7B4C   F0 04      BEQ $7B52
7B4E   C9 79      CMP #$79
7B50   D0 2A      BNE $7B7C     Nein, Skip
7B52   A9 D2      LDA #$D2      Zeiger auf Interpreterwert $03D2
7B54   A0 03      LDY #$03
7B56   60         RTS

7B57   C0 C9      CPY #$C9      TI$ ?
7B59   F0 F7      BEQ $7B52     Ja, Zeiger auf Interpreterwert setzen
7B5B   C0 49      CPY #$49      TI ?
7B5D   D0 31      BNE $7B90     Nein, Skip
7B5F   F0 18      BEQ $7B79     Ja, Fehler

7B61   C0 D3      CPY #$D3      DS$ ?
7B63   F0 14      BEQ $7B79     Ja, Fehler
7B65   C0 53      CPY #$53      DS ?
7B67   D0 27      BNE $7B90     Nein, Skip
7B69   F0 0E      BEQ $7B79     Ja, Fehler

7B6B   C0 54      CPY #$54      ST ?
7B6D   D0 21      BNE $7B90     Nein, Skip
```

```
7B6F   F0 08      BEQ $7B79   Ja, Fehler

7B71   C0 52      CPY #$52    ER ?
7B73   F0 04      BEQ $7B79   Ja, Fehler
7B75   C0 4C      CPY #$4C    EL ?
7B77   D0 17      BNE $7B90   Nein, Skip
7B79   4C 6C 79   JMP $796C   'SYNTAX'

7B7C   A5 47      LDA $47     Variablenname laden
7B7E   A4 48      LDY $48
7B80   C9 54      CMP #$54    TI oder TI$ ?
7B82   F0 D3      BEQ $7B57   Ja, testen
7B84   C9 53      CMP #$53    ST ?
7B86   F0 E3      BEQ $7B6B   Ja, testen
7B88   C9 45      CMP #$45    ER oder EL ?
7B8A   F0 E5      BEQ $7B71   Ja, testen
7B8C   C9 44      CMP #$44    DS oder DS$ ?
7B8E   F0 D1      BEQ $7B61   Ja, testen
7B90   A5 31      LDA $31     Zeiger auf Arraystart
7B92   A4 32      LDY $32
7B94   85 61      STA $61     retten
7B96   84 62      STY $62
7B98   A5 33      LDA $33     Zeiger auf Arrayende
7B9A   A4 34      LDY $34
7B9C   85 5C      STA $5C     retten
7B9E   84 5D      STY $5D
7BA0   18         CLC
7BA1   69 07      ADC #$07
7BA3   90 01      BCC $7BA6   Zeiger für neue Variable um 7 erhöhen
7BA5   C8         INY
7BA6   85 5A      STA $5A
7BA8   84 5B      STY $5B
7BAA   20 66 7C   JSR $7C66   Block verschieben
7BAD   A5 5A      LDA $5A
7BAF   A4 5B      LDY $5B
7BB1   C8         INY
7BB2   85 31      STA $31     Zeiger auf Arraystart neu setzen
7BB4   84 32      STY $32
7BB6   85 5A      STA $5A
7BB8   84 5B      STY $5B
7BBA   A5 5A      LDA $5A     Stringtrailer müssen korrigiert werden
7BBC   A6 5B      LDX $5B
7BBE   E4 34      CPX $34     Ende erreicht ?
7BC0   D0 06      BNE $7BC8
7BC2   C5 33      CMP $33
7BC4   D0 02      BNE $7BC8
7BC6   F0 78      BEQ $7C40   Ja, Variable eintragen
7BC8   85 24      STA $24     Zeiger setzen
```

```
7BCA   86 25      STX $25
7BCC   A0 00      LDY #$00
7BCE   20 B7 03   JSR $03B7    Erstes Zeichen des Namens
7BD1   AA         TAX          in (X)
7BD2   C8         INY
7BD3   20 B7 03   JSR $03B7    Zweites Zeichen des Namens laden
7BD6   08         PHP          und Status auf Stapel (BPL oder BMI !)
7BD7   C8         INY
7BD8   20 B7 03   JSR $03B7    Feldlänge auf Zeiger aufaddieren
7BDB   65 5A      ADC $5A
7BDD   85 5A      STA $5A
7BDF   C8         INY
7BE0   20 B7 03   JSR $03B7
7BE3   65 5B      ADC $5B
7BE5   85 5B      STA $5B
7BE7   28         PLP          Stringvariable ?
7BE8   10 D0      BPL $7BBA    Nein, zum Schleifenstart
7BEA   8A         TXA
7BEB   30 CD      BMI $7BBA    Nein, zum Schleifenstart
7BED   C8         INY
7BEE   20 B7 03   JSR $03B7    Dimensionen laden
7BF1   A0 00      LDY #$00
7BF3   0A         ASL          * 2
7BF4   69 05      ADC #$05     + 5
7BF6   65 24      ADC $24      ergibt Zeiger auf Feldanfang
7BF8   85 24      STA $24
7BFA   90 02      BCC $7BFE
7BFC   E6 25      INC $25
7BFE   A6 25      LDX $25      Feldende erreicht ?
7C00   E4 5B      CPX $5B
7C02   D0 04      BNE $7C08
7C04   C5 5A      CMP $5A
7C06   F0 B6      BEQ $7BBE    Ja, zum Schleifenstart
7C08   A0 00      LDY #$00
7C0A   20 B7 03   JSR $03B7    Länge des Strings holen
7C0D   F0 24      BEQ $7C33    Wenn 0, Skip
7C0F   85 79      STA $79      Länge setzen
7C11   C8         INY
7C12   20 B7 03   JSR $03B7    Stringadresse
7C15   18         CLC
7C16   65 79      ADC $79      + Länge
7C18   85 5C      STA $5C
7C1A   C8         INY
7C1B   20 B7 03   JSR $03B7
7C1E   69 00      ADC #$00
7C20   85 5D      STA $5D      ergibt Zeiger auf Trailer
7C22   A0 00      LDY #$00
7C24   20 E2 42   JSR $42E2    Trailer um 7 erhöhen
```

```
7C27   69 07      ADC #$07
7C29   91 5C      STA ($5C),Y
7C2B   C8         INY
7C2C   20 E2 42   JSR $42E2
7C2F   69 00      ADC #$00
7C31   91 5C      STA ($5C),Y
7C33   A9 03      LDA #$03      Zeiger auf nächstes Arrayelement setzen
7C35   18         CLC
7C36   65 24      ADC $24
7C38   85 24      STA $24
7C3A   90 C2      BCC $7BFE
7C3C   E6 25      INC $25
7C3E   D0 BE      BNE $7BFE     und Array weiter testen

7C40   A0 00      LDY #$00      Offset auf
7C42   A5 47      LDA $47       Namen eintragen
7C44   8D 04 FF   STA $FF04     Write in Bank 1 setzen
7C47   91 61      STA ($61),Y
7C49   C8         INY
7C4A   A5 48      LDA $48
7C4C   91 61      STA ($61),Y
7C4E   A9 00      LDA #$00
7C50   C8         INY
7C51   91 61      STA ($61),Y   Variablenwert auf 0 setzen
7C53   C0 06      CPY #$06
7C55   D0 F9      BNE $7C50
7C57   A5 61      LDA $61       Variablenstart
7C59   18         CLC
7C5A   69 02      ADC #$02      + 2
7C5C   A4 62      LDY $62
7C5E   90 01      BCC $7C61
7C60   C8         INY
7C61   85 49      STA $49       = Zeiger auf Variablenwert
7C63   84 4A      STY $4A
7C65   60         RTS

******************************  Blockverschieberoutine für Variablen

                                ($61) = Alter Blockanfang
                                ($5C) = Altes Blockende + 1
                                ($5A) = Neues Blockende + 1

7C66   20 17 50   JSR $5017     Im Speicher Platz schaffen
7C69   85 33      STA $33       (A) und (Y) speichern
7C6B   84 34      STY $34
7C6D   38         SEC
7C6E   A5 5C      LDA $5C       Altes Blockende
7C70   E5 61      SBC $61       - Alter Blockanfang + 1
```

```
7C72   85 24     STA $24
7C74   A8        TAY          ergibt Restabschnitt
7C75   A5 5D     LDA $5D
7C77   E5 62     SBC $62
7C79   AA        TAX          und Anzahl der Speicherseiten
7C7A   E8        INX          die verschoben werden müssen
7C7B   98        TYA          Restabschnitt vorhanden ?
7C7C   F0 25     BEQ $7CA3    Nein, Skip
7C7E   A5 5C     LDA $5C      Zeiger auf Restabschnitt anpassen
7C80   38        SEC
7C81   E5 24     SBC $24
7C83   85 5C     STA $5C
7C85   B0 03     BCS $7C8A
7C87   C6 5D     DEC $5D
7C89   38        SEC
7C8A   A5 5A     LDA $5A
7C8C   E5 24     SBC $24
7C8E   85 5A     STA $5A
7C90   B0 09     BCS $7C9B
7C92   C6 5B     DEC $5B
7C94   90 05     BCC $7C9B    und Restabschnitt zuerst kopieren

7C96   20 E2 42  JSR $42E2    LDA ($5C),Y
7C99   91 5A     STA ($5A),Y  Block zeichenweise übertragen
7C9B   88        DEY
7C9C   D0 F8     BNE $7C96
7C9E   20 E2 42  JSR $42E2    Letztes Zeichen in der Seite übertragen
7CA1   91 5A     STA ($5A),Y
7CA3   C6 5D     DEC $5D      Seitenzeiger erniedrigen
7CA5   C6 5B     DEC $5B
7CA7   CA        DEX          Alle Seiten übertragen ?
7CA8   D0 F1     BNE $7C9B    Nein, weitersuchen
7CAA   60        RTS

******************************* Dimensionierte Variable holen

7CAB   A5 0E     LDA $0E      DIM-Flag und
7CAD   05 10     ORA $10      Integerflag verknüpfen
7CAF   48        PHA          und retten
7CB0   A5 0F     LDA $0F      Typflag retten
7CB2   48        PHA
7CB3   A0 00     LDY #$00     Anzahl der Indizes auf 0
7CB5   98        TYA
7CB6   48        PHA          und retten
7CB7   A5 48     LDA $48      Variablenname retten
7CB9   48        PHA
7CBA   A5 47     LDA $47
7CBC   48        PHA
```

```
7CBD   20 A7 84   JSR $84A7     Index laden und in Integer
7CC0   68         PLA
7CC1   85 47      STA $47       Variablennamen wieder setzen
7CC3   68         PLA
7CC4   85 48      STA $48
7CC6   68         PLA
7CC7   A8         TAY           Anzahl der Indizes holen
7CC8   BA         TSX
7CC9   BD 02 01   LDA $0102,X   Ersten Wert auf Stapel verdoppeln
7CCC   48         PHA
7CCD   BD 01 01   LDA $0101,X
7CD0   48         PHA
7CD1   A5 66      LDA $66       und Index auf Stapel
7CD3   9D 02 01   STA $0102,X
7CD6   A5 67      LDA $67
7CD8   9D 01 01   STA $0101,X
7CDB   C8         INY           Anzahl der Indizes erhöhen
7CDC   84 0D      STY $0D       und setzen
7CDE   20 86 03   JSR $0386     CHRGOT
7CE1   A4 0D      LDY $0D       Anzahl wieder laden
7CE3   C9 2C      CMP #$2C      Zeichen = ',' ?
7CE5   F0 CE      BEQ $7CB5     Ja, nächsten Index verarbeiten
7CE7   20 56 79   JSR $7956     Test auf ')'
7CEA   68         PLA
7CEB   85 0F      STA $0F       Typflag wieder setzen
7CED   68         PLA
7CEE   85 10      STA $10       Integerflag wieder setzen
7CF0   29 7F      AND #$7F
7CF2   85 0E      STA $0E       und DIM-Wert setzen
7CF4   A6 31      LDX $31       Zeiger auf Arraystart
7CF6   A5 32      LDA $32
7CF8   86 61      STX $61       in Hilfszeiger setzen
7CFA   85 62      STA $62
7CFC   C5 34      CMP $34       Ende erreicht ?
7CFE   D0 04      BNE $7D04
7D00   E4 33      CPX $33
7D02   F0 42      BEQ $7D46     Ja, Skip
7D04   A0 00      LDY #$00
7D06   20 00 43   JSR $4300     Variablenname gefunden ?
7D09   C8         INY
7D0A   C5 47      CMP $47
7D0C   D0 07      BNE $7D15
7D0E   20 00 43   JSR $4300
7D11   C5 48      CMP $48
7D13   F0 18      BEQ $7D2D     Ja, Skip
7D15   C8         INY
7D16   20 00 43   JSR $4300     Arraylänge auf Zeiger aufaddieren
7D19   18         CLC
```

```
7D1A   65 61      ADC $61
7D1C   AA         TAX
7D1D   C8         INY
7D1E   20 00 43   JSR $4300
7D21   65 62      ADC $62
7D23   90 D3      BCC $7CF8    und weitersuchen

7D25   A2 12      LDX #$12     'BAD SUBSCRIPT'
7D27   2C         .BYTE $2C
7D28   A2 0E      LDX #$0E     'ILLEGAL QUANTITY'
7D2A   4C 3C 4D   JMP $4D3C

7D2D   A2 13      LDX #$13     Wert für 'REDIM'D ARRAY'
7D2F   A5 0E      LDA $0E      DIM-Flag gesetzt ?
7D31   D0 F7      BNE $7D2A    Ja, Fehler
7D33   20 71 7E   JSR $7E71    Zeiger auf erstes Element berechnen
7D36   A0 04      LDY #$04
7D38   20 00 43   JSR $4300    Dimensionen aus Array laden
7D3B   85 79      STA $79
7D3D   A5 0D      LDA $0D      Gesuchte Dimensionen
7D3F   C5 79      CMP $79      = gefundene Dimensionen ?
7D41   D0 E2      BNE $7D25    Nein, Fehler
7D43   4C D2 7D   JMP $7DD2    Gesuchtes Arrayelement anwählen

****************************** Dimensionierte Variable neu anlegen

7D46   20 71 7E   JSR $7E71    Zeiger auf erstes Element berechnen
7D49   20 17 50   JSR $5017    Platz im Speicher schaffen
7D4C   A0 00      LDY #$00     Offset auf 0
7D4E   84 73      STY $73
7D50   A2 05      LDX #$05     Elementlänge auf 5 für Real
7D52   A5 47      LDA $47      Erstes Zeichen des Namens
7D54   8D 04 FF   STA $FF04    Write in Bank 1 setzen
7D57   91 61      STA ($61),Y  Zeichen in Array setzen
7D59   10 01      BPL $7D5C    Wenn kein Integer, Skip
7D5B   CA         DEX          (X) auf 4
7D5C   C8         INY
7D5D   A5 48      LDA $48      Zweites Zeichen des Namens setzen
7D5F   91 61      STA ($61),Y
7D61   10 02      BPL $7D65    Wenn Real-Variable, Skip
7D63   CA         DEX          (X) auf 2 für Integer
7D64   CA         DEX          und auf 3 für String
7D65   86 72      STX $72      Elementlänge setzen
7D67   A5 0D      LDA $0D      Dimensionsanzahl
7D69   C8         INY
7D6A   C8         INY
7D6B   C8         INY
7D6C   91 61      STA ($61),Y  in Array setzen
```

```
7D6E   A2 0B      LDX #$0B
7D70   A9 00      LDA #$00
7D72   24 0E      BIT $0E        DIM-Befehl ?
7D74   50 08      BVC $7D7E      Nein, Skip
7D76   68         PLA            Dimension vom Stack holen
7D77   18         CLC
7D78   69 01      ADC #$01       um 1 erhöhen
7D7A   AA         TAX
7D7B   68         PLA
7D7C   69 00      ADC #$00       High-Byte korrigieren
7D7E   C8         INY
7D7F   91 61      STA ($61),Y    und Dimension in Array speichern
7D81   C8         INY
7D82   8A         TXA
7D83   91 61      STA ($61),Y
7D85   20 3E 7E   JSR $7E3E      Platz für restliche Dimensionen berechnen
7D88   86 72      STX $72        und Endezeiger setzen
7D8A   85 73      STA $73
7D8C   A4 24      LDY $24        Offset wieder laden (gesetzt JSR $7E3E)
7D8E   C6 0D      DEC $0D        Alle Dimensionen verarbeitet ?
7D90   D0 DC      BNE $7D6E      Nein, weiter eintragen
7D92   65 5B      ADC $5B        Feldlänge auf Start aufaddieren
7D94   B0 67      BCS $7DFD
7D96   85 5B      STA $5B
7D98   A8         TAY
7D99   8A         TXA
7D9A   65 5A      ADC $5A
7D9C   90 03      BCC $7DA1
7D9E   C8         INY
7D9F   F0 5C      BEQ $7DFD      Bei Übertrag, Fehler
7DA1   20 17 50   JSR $5017      und Speicherplatz prüfen
7DA4   85 33      STA $33        Feldendezeiger neu setzen
7DA6   84 34      STY $34
7DA8   A9 00      LDA #$00       Array mit Nullbytes füllen
7DAA   E6 73      INC $73
7DAC   A4 72      LDY $72
7DAE   F0 05      BEQ $7DB5
7DB0   88         DEY
7DB1   91 5A      STA ($5A),Y
7DB3   D0 FB      BNE $7DB0
7DB5   C6 5B      DEC $5B
7DB7   C6 73      DEC $73        Alles gefüllt ?
7DB9   D0 F5      BNE $7DB0      Nein, weitermachen

7DBB   E6 5B      INC $5B
7DBD   38         SEC
7DBE   A5 33      LDA $33
```

```
7DC0   E5 61      SBC $61
7DC2   A0 02      LDY #$02
7DC4   91 61      STA ($61),Y  Feldlänge in Array setzen
7DC6   A5 34      LDA $34
7DC8   C8         INY
7DC9   E5 62      SBC $62
7DCB   91 61      STA ($61),Y
7DCD   A5 0E      LDA $0E      DIM-Befehl ?
7DCF   D0 6C      BNE $7E3D    Ja, Ende

******************************** Arrayelement suchen

7DD1   C8         INY
7DD2   20 00 43   JSR $4300    Anzahl der Dimensionen
7DD5   85 0D      STA $0D      setzen
7DD7   A9 00      LDA #$00
7DD9   85 72      STA $72
7DDB   85 73      STA $73
7DDD   C8         INY
7DDE   68         PLA          Index vom Stapel holen
7DDF   AA         TAX
7DE0   85 66      STA $66
7DE2   20 00 43   JSR $4300    Entsprechenden Wert aus Array holen
7DE5   85 79      STA $79
7DE7   68         PLA
7DE8   85 67      STA $67
7DEA   C5 79      CMP $79      Index zu groß ?
7DEC   90 12      BCC $7E00    Nein, Skip
7DEE   D0 0A      BNE $7DFA    Ja, Fehler
7DF0   C8         INY
7DF1   20 00 43   JSR $4300    Index zu groß ?
7DF4   85 79      STA $79
7DF6   E4 79      CPX $79
7DF8   90 07      BCC $7E01    Nein, Skip
7DFA   4C 25 7D   JMP $7D25    'BAD SUBSCRIPT'
7DFD   4C 3A 4D   JMP $4D3A    'OUT OF MEMORY'

******************************** Elementadresse berechnen

7E00   C8         INY
7E01   A5 73      LDA $73      (X)/(A) = ($72) * (($66),Y)
7E03   05 72      ORA $72
7E05   18         CLC
7E06   F0 0A      BEQ $7E12
7E08   20 3E 7E   JSR $7E3E    Multiplikationsroutine
7E0B   8A         TXA
7E0C   65 66      ADC $66
7E0E   AA         TAX
```

```
7E0F   98         TYA
7E10   A4 24      LDY $24       Offset wieder laden
7E12   65 67      ADC $67
7E14   86 72      STX $72
7E16   C6 0D      DEC $0D       Noch eine Dimension ?
7E18   D0 C1      BNE $7DDB     Ja, die auch verarbeiten
7E1A   85 73      STA $73
7E1C   A2 05      LDX #$05      Wert für Real-Berechnung
7E1E   A5 47      LDA $47       Integervariable ?
7E20   10 01      BPL $7E23     Nein, Skip
7E22   CA         DEX           (X) = 4
7E23   A5 48      LDA $48       Realvariable ?
7E25   10 02      BPL $7E29     Ja, Skip
7E27   CA         DEX           (X) = 2 für Integer
7E28   CA         DEX           (X) = 3 für String
7E29   86 2A      STX $2A       Low-Byte für Multiplikation setzen
7E2B   A9 00      LDA #$00      High-Byte auf 0
7E2D   20 49 7E   JSR $7E49     Multiplikationsroutine
7E30   8A         TXA           Ergebnis
7E31   65 5A      ADC $5A       + Feldstart
7E33   85 49      STA $49       ergibt Variablenzeiger
7E35   98         TYA
7E36   65 5B      ADC $5B
7E38   85 4A      STA $4A
7E3A   A8         TAY
7E3B   A5 49      LDA $49       Wert auch in (A)/(Y)
7E3D   60         RTS

******************************** Multiplikationsroutine

7E3E   84 24      STY $24       Offset retten
7E40   20 00 43   JSR $4300     Dimension aus Array holen und setzen
7E43   85 2A      STA $2A
7E45   88         DEY
7E46   20 00 43   JSR $4300

7E49   85 2B      STA $2B

******************************** (X)/(Y) = ($72) * ($2A)

7E4B   A9 10      LDA #$10      16 Bit multiplizieren
7E4D   85 5F      STA $5F
7E4F   A2 00      LDX #$00      Ergebnis auf 0 setzen
7E51   A0 00      LDY #$00
7E53   8A         TXA           Ergebnis * 2
7E54   0A         ASL
7E55   AA         TAX
7E56   98         TYA
```

```
7E57   2A        ROL
7E58   A8        TAY
7E59   B0 A2     BCS $7DFD    Wenn Übertrag, Fehler
7E5B   06 72     ASL $72      Multiplikand * 2
7E5D   26 73     ROL $73
7E5F   90 0B     BCC $7E6C    Wenn kein Übertrag, Skip
7E61   18        CLC          Multiplikator zu Ergebnis addieren
7E62   8A        TXA
7E63   65 2A     ADC $2A
7E65   AA        TAX
7E66   98        TYA
7E67   65 2B     ADC $2B
7E69   A8        TAY
7E6A   B0 91     BCS $7DFD    Wenn Übertrag, Fehler
7E6C   C6 5F     DEC $5F      Alle Bits verarbeitet ?
7E6E   D0 E3     BNE $7E53    Nein, weitermachen
7E70   60        RTS

****************************** Zeiger auf erstes Arrayelement berechnen

7E71   A5 0D     LDA $0D      Anzahl der Dimensionen
7E73   0A        ASL          * 2
7E74   69 05     ADC #$05     + Länge des Feldkopfes
7E76   65 61     ADC $61      + Feldadresse
7E78   A4 62     LDY $62
7E7A   90 01     BCC $7E7D
7E7C   C8        INY
7E7D   85 5A     STA $5A      ergibt Elementadresse
7E7F   84 5B     STY $5B
7E81   60        RTS

****************************** Füllbytes ohne Bedeutung

7E82   FF FF FF FF . . .
7FFE   . . . FF FF
```

```
****************************** BASIC-Funktion FRE

8000   20 F7 87   JSR $87F7    Byte-Wert holen
8003   E0 01      CPX #$01     Bank 1 ?
8005   90 05      BCC $800C    Nein, Bank 0 Skip
8007   F0 31      BEQ $803A    Wenn Bank 1, Skip
8009   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'
800C   38         SEC          (A)/(Y) = Bank 0 Maximal - Programmende
800D   AD 12 12   LDA $1212    Low-Byte Bank 0 Maximal
8010   ED 10 12   SBC $1210    - Low-Byte Programmende
8013   A8         TAY          in (Y) retten
8014   AD 13 12   LDA $1213    High-Byte Bank 0 Maximal
8017   ED 11 12   SBC $1211    - High-Byte Programmende
801A   B0 2B      BCS $8047    Wenn gültig, Wert in FAC#1

801C   A6 35      LDX $35      Probieren Sie SYS32800,123,45,6 aus !
801E   E8         INX          Das Ergebnis wird Sie überaschen !
801F   C8         INY
8020   85 70      STA $70
8022   98         TYA
8023   38         SEC
8024   E9 05      SBC #$05
8026   85 71      STA $71
8028   A5 71      LDA $71
802A   5D 37 AE   EOR $AE37,X
802D   45 70      EOR $70
802F   F0 44      BEQ $8075    Wenn Ende, Skip
8031   20 D2 FF   JSR $FFD2    Zeichen ausgeben
8034   E6 71      INC $71
8036   E8         INX
8037   D0 EF      BNE $8028    Unbedingter Sprung
8039   60         RTS

803A   20 EA 92   JSR $92EA    Garbage-Collection
803D   38         SEC          (A)/(Y) = Stringanfang - Variablenende
803E   A5 35      LDA $35      Low-Byte Stringanfang
8040   E5 33      SBC $33      - Low-Byte Variablenende
8042   A8         TAY          in (Y) retten
8043   A5 36      LDA $36      High-Byte Stringanfang
8045   E5 34      SBC $34      - High-Byte Variablenende
8047   4C C9 84   JMP $84C9    (A)/(Y) in FAC#1

****************************** BASIC-Funktion VAL

804A   20 6E 86   JSR $866E    Stringparameter holen
804D   D0 03      BNE $8052    Wenn Länge > 0, Skip
804F   4C D6 88   JMP $88D6    FAC#1 = 0
8052   18         CLC
```

```
8053   65 24      ADC $24       Adresse Low-Byte zu Länge addieren
8055   85 72      STA $72       Zeiger auf Ende (Trailer) setzen
8057   A5 25      LDA $25       Adresse High-Byte laden
8059   69 00      ADC #$00      Übertrag berücksichtigen
805B   85 73      STA $73       und Traileradresse High-Byte setzen
805D   A0 00      LDY #$00      Offset auf 0
805F   A9 72      LDA #$72      Ergit LDA ($72) aus Bank 1
8061   20 AB 03   JSR $03AB     Erstes Trailerbyte holen
8064   48         PHA           und retten
8065   98         TYA           (A) = 0
8066   91 72      STA ($72),Y   Byte durch 0 als Ende ersetzen
8068   20 03 8E   JSR $8E03     CHRGOT aus ($24)
806B   A2 01      LDX #$01      Flag für Bank 1
806D   20 22 8D   JSR $8D22     ASCII-Zahl aus Bank 1 in FAC#1
8070   68         PLA           Trailerbyte wieder holen
8071   A0 00      LDY #$00      Offset auf 0
8073   91 72      STA ($72),Y   Trailerbyte wieder setzen
8075   60         RTS

****************************** BASIC-Funktion DEC

8076   20 6E 86   JSR $866E     Stringparameter holen
8079   85 26      STA $26       Länge setzen
807B   A0 00      LDY #$00      Offset auf 0
807D   84 27      STY $27       Stellenzähler löschen
807F   84 72      STY $72       Wert Low-Byte löschen
8081   84 73      STY $73       Wert High-Byte löschen
8083   C4 26      CPY $26       Stringende erreicht ?
8085   F0 34      BEQ $80BB     Ja, Skip
8087   20 B7 03   JSR $03B7     Zeichen aus String holen
808A   C8         INY           Offset erhöhen
808B   C9 20      CMP #$20      Space ?
808D   F0 F4      BEQ $8083     Ja, überlesen
808F   E6 27      INC $27       1 Stelle mehr
8091   A6 27      LDX $27       Stellenzähler laden
8093   E0 05      CPX #$05      5 Stellen erreicht ?
8095   F0 2B      BEQ $80C2     Ja, Fehler
8097   C9 30      CMP #$30      < '0' ?
8099   90 27      BCC $80C2     Ja, Fehler
809B   C9 3A      CMP #$3A      <= '9' ?
809D   90 0A      BCC $80A9     Ja, Skip
809F   C9 41      CMP #$41      < 'A' ?
80A1   90 1F      BCC $80C2     Ja, Fehler
80A3   C9 47      CMP #$47      > 'F' ?
80A5   B0 1B      BCS $80C2     Ja, Fehler
80A7   E9 07      SBC #$07      Buchstaben zu Zahlen
80A9   E9 2F      SBC #$2F      ASCII zu Zahl umwandeln
80AB   0A         ASL           Wert = Wert * 16
```

```
80AC    0A          ASL
80AD    0A          ASL
80AE    0A          ASL
80AF    A2 04       LDX #$04     Wert in Ergebnis setzen
80B1    0A          ASL          Bit aus Wert ins C-Flag schieben
80B2    26 72       ROL $72      C-Flag ins Ergebnis
80B4    26 73       ROL $73      übernehmen
80B6    CA          DEX          Alle 4 Bits in Ergebnis gesetzt ?
80B7    D0 F8       BNE $80B1    Nein, weitermachen
80B9    F0 C8       BEQ $8083    Nächstes Zeichen holen

80BB    A4 72       LDY $72      Wert Low-Byte laden
80BD    A5 73       LDA $73      Wert High-Byte laden
80BF    4C C9 84    JMP $84C9    Adresse in FAC#1
80C2    4C 28 7D    JMP $7D28    'ILLEGAL QUANTITY'

****************************** BASIC-Funktion PEEK

80C5    A5 17       LDA $17      Adresse High-Byte
80C7    48          PHA          retten
80C8    A5 16       LDA $16      Adresse Low-Byte
80CA    48          PHA          retten
80CB    20 DA 77    JSR $77DA    PEEK-Adresse holen
80CE    20 15 88    JSR $8815    in Adreßformat
80D1    AE D5 03    LDX $03D5    Bank laden
80D4    A0 00       LDY #$00     Offset laden
80D6    A9 16       LDA #$16     Ergibt LDA ($16),Y
80D8    20 74 FF    JSR $FF74    LDA ($16),Y aus Bank (X)
80DB    A8          TAY          Wert retten
80DC    68          PLA          Adresse wieder holen
80DD    85 16       STA $16      und Low-Byte setzen
80DF    68          PLA          High-Byte holen
80E0    85 17       STA $17      und setzen
80E2    4C D4 84    JMP $84D4    Byte-Wert in FAC#1

****************************** BASIC-Befehl POKE

80E5    20 03 88    JSR $8803    POKE-Adresse und Wert holen
80E8    8A          TXA          Wert in (A)
80E9    A0 00       LDY #$00     Offset setzen
80EB    A2 16       LDX #$16     Ergibt STA ($16),Y
80ED    8E B9 02    STX $02B9    Adresse ($16) wählen
80F0    AE D5 03    LDX $03D5    Bank setzen
80F3    4C 77 FF    JMP $FF77    STA ($16),Y in Bank (X)

****************************** BASIC-Funktion ERR$

80F6    20 F7 87    JSR $87F7    Byte-Wert holen
```

```
80F9  CA        DEX          Wert anpassen (1-41)
80FA  8A        TXA
80FB  C9 29     CMP #$29     > 41 ?
80FD  B0 37     BCS $8136    Ja, Fehler
80FF  20 82 4A  JSR $4A82    Fehlermeldungsadresse setzen
8102  A0 FF     LDY #$FF     Offset auf Start
8104  A2 00     LDX #$00     Länge auf 0
8106  E8        INX          Länge erhöhen
8107  C8        INY          Offset erhöhen
8108  B1 26     LDA ($26),Y  Ende erreicht ?
810A  30 06     BMI $8112    Ja, Skip
810C  C9 20     CMP #$20     Code < Space ? (Sondercode)
810E  90 F7     BCC $8107    Ja, überlesen
8110  B0 F4     BCS $8106    Weiterschleifen
8112  8A        TXA          Länge in (A)
8113  20 90 86  JSR $8690    Platz für (A) Bytes reservieren
8116  A2 00     LDX #$00     Fehlertext in Variablenbank übertragen
8118  A0 FF     LDY #$FF     Offset auf Start
811A  8D 04 FF  STA $FF04    Read aus Bank 1 setzen
811D  C8        INY          Offset erhöhen
811E  B1 26     LDA ($26),Y  Fehlertextbyte holen
8120  C9 20     CMP #$20     Code < Space ? (Sondercode)
8122  90 F9     BCC $811D    Ja, überlesen
8124  20 39 81  JSR $8139    (X) und (Y) vertauschen
8127  48        PHA          Zeichen retten
8128  29 7F     AND #$7F     RVS-Bit löschen
812A  91 64     STA ($64),Y  und in String kopieren
812C  20 39 81  JSR $8139    (X) und (Y) vertauschen
812F  E8        INX          Länge erhöhen
8130  68        PLA          Ende erreicht ?
8131  10 EA     BPL $811D    Nein, weitermachen
8133  4C D1 85  JMP $85D1    Stringparameter setzen
8136  4C 28 7D  JMP $7D28    'ILLEGAL QUANTITY'

****************************** (X) <=> (Y)

8139  48        PHA          (A) retten
813A  8A        TXA          (X) auf Stapel
813B  48        PHA          retten
813C  98        TYA          (Y) in (A)
813D  AA        TAX          und dann in (X)
813E  68        PLA          altes (X)
813F  A8        TAY          ergibt neues (Y)
8140  68        PLA          (A) wieder holen
8141  60        RTS
```

```
****************************** BASIC-Funktion HEX$

8142   20 DA 77   JSR $77DA    Auf numerisch prüfen
8145   A5 16      LDA $16      Adresse Low-Byte laden
8147   48         PHA          und retten
8148   A5 17      LDA $17      Adresse High-Byte laden
814A   48         PHA          und retten
814B   20 15 88   JSR $8815    FAC#1 in Adreßformat
814E   A9 04      LDA #$04     Die Hexzahl hat 4 Stellen
8150   20 90 86   JSR $8690    4 Bytes im Stringbereich reservieren
8153   A0 00      LDY #$00     Offset auf 0
8155   A5 17      LDA $17      Adresse High-Byte laden
8157   8D 04 FF   STA $FF04    Write in Bank 1 setzen
815A   20 6B 81   JSR $816B    High-Byte in String
815D   A5 16      LDA $16      Adresse Low-Byte laden
815F   20 6B 81   JSR $816B    Low-Byte in String
8162   68         PLA          Adresse High-Byte wieder holen
8163   85 17      STA $17      und setzen
8165   68         PLA          Adresse Low-Byte wieder holen
8166   85 16      STA $16      und setzen
8168   4C D1 85   JMP $85D1    Stringparameter setzen

****************************** Byte in HEX-ASCII ab ($64),Y

816B   48         PHA          Bytewert retten
816C   4A         LSR          Oberes Nibble in
816D   4A         LSR          Unteres Nibble schieben
816E   4A         LSR
816F   4A         LSR
8170   20 74 81   JSR $8174    Nibble umwandeln
8173   68         PLA          Bytewert wieder holen

****************************** Nibble in HEX-ASCII in ($64),Y

8174   29 0F      AND #$0F     Nibble-Wert isolieren
8176   C9 0A      CMP #$0A     Ziffer ?
8178   90 02      BCC $817C    Ja, Skip
817A   69 06      ADC #$06     Wert in Buchstabencode wandeln
817C   69 30      ADC #$30     Ziffer in ASCII umrechnen
817E   91 64      STA ($64),Y  ASCII-Zeichen in String
8180   C8         INY          Offset erhöhen
8181   60         RTS

****************************** BASIC-Funktion RGR

8182   20 DA 77   JSR $77DA    Auf numerisch prüfen
8185   20 8C 81   JSR $818C    Graphikmodus laden
8188   A8         TAY          Byte-Wert in FAC#1
```

```
8189   4C D4 84   JMP $84D4    setzen

****************************** Aktiven Graphikmodus in (A) laden

818C   A5 D8      LDA $D8      Modusflag laden
818E   18         CLC          Bit 6 in Bit 0
818F   2A         ROL          und
8190   2A         ROL          Bit 7 in Bit 1 bringen
8191   2A         ROL
8192   69 00      ADC #$00     und Bit 5 dazuaddieren
8194   24 D7      BIT $D7      80-Zeichen-Modus ?
8196   10 02      BPL $819A    Nein, Skip
8198   69 05      ADC #$05     5 für aktiven 80-Zeichen-Schirm addieren
819A   60         RTS

****************************** BASIC-Funktion RCLR

819B   20 F7 87   JSR $87F7    FAC#1 in Byte-Wert in (X)
819E   20 45 A8   JSR $A845    ROMs einschalten
81A1   CA         DEX          40-Zeichen-Hintergrund ?
81A2   30 15      BMI $81B9    Ja, Skip
81A4   CA         DEX          Vordergrund ?
81A5   30 1A      BMI $81C1    Ja, Skip
81A7   CA         DEX          Mehrfarbe 1 ?
81A8   30 1C      BMI $81C6    Ja, Skip
81AA   CA         DEX          Mehrfarbe 2 ?
81AB   30 1E      BMI $81CB    Ja, Skip
81AD   CA         DEX          40-Zeichen-Rand ?
81AE   30 20      BMI $81D0    Ja, Skip
81B0   CA         DEX          80-Zeichen-Text ?
81B1   30 23      BMI $81D6    Ja, Skip
81B3   CA         DEX          80-Zeichen-Hintergrund ?
81B4   30 28      BMI $81DE    Ja, Skip
81B6   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'
81B9   AD 21 D0   LDA $D021    40-Zeichen-Hintergrund
81BC   29 7F      AND #$7F     Farbwert
81BE   4C EC 81   JMP $81EC    Wert setzen
81C1   A5 86      LDA $86      Vordergrund Farbwert
81C3   4C EC 81   JMP $81EC    Wert setzen
81C6   A5 84      LDA $84      Mehrfarbwert 1
81C8   4C EC 81   JMP $81EC    Wert setzen
81CB   A5 85      LDA $85      Mehrfarbwert 2
81CD   4C EC 81   JMP $81EC    Wert setzen
81D0   AD 20 D0   LDA $D020    40-Zeichen-Rand Farbwert
81D3   4C EC 81   JMP $81EC    Wert setzen
81D6   A5 F1      LDA $F1      80-Zeichen-Text Farbwert
81D8   24 D7      BIT $D7      80-Zeichen ?
81DA   10 10      BPL $81EC    Nein, Skip
```

```
81DC   30 08      BMI $81E6    Ja, Farbwert umwandeln
81DE   A9 1A      LDA #$1A     80-Zeichen-Hintergrund
81E0   8D 00 D6   STA $D600    Registernummer programmieren
81E3   AD 01 D6   LDA $D601    Farbwert einlesen
81E6   29 0F      AND #$0F     Farbnibble isolieren
81E8   AA         TAX          und als Offset laden
81E9   BD F3 81   LDA $81F3,X  Farbwerttabelle 80-Zeichen
81EC   29 0F      AND #$0F     Farbwert isolieren
81EE   A8         TAY
81EF   C8         INY          und in Bereich von 1 bis 16 bringen
81F0   4C D4 84   JMP $84D4    Byte-Wert in FAC#1

****************************** Farbwerttabelle für 80-Zeichen

81F3   00 0C 06 0E 05 0D 0B 03
81FB   02 0A 08 04 09 07 0F 01

****************************** BASIC-Funktion JOY

8203   20 F7 87   JSR $87F7    Byte-Wert holen
8206   CA         DEX          Wert anpassen (1 oder 2)
8207   E0 02      CPX #$02     > 2 ?
8209   B0 34      BCS $823F    Ja, Fehler
820B   8A         TXA          Wert invertieren
820C   49 01      EOR #$01
820E   AA         TAX
820F   08         PHP          Status retten
8210   20 45 A8   JSR $A845    ROMs einschalten
8213   78         SEI          Tastaturabfrage verbieten
8214   AD 00 DC   LDA $DC00    Port A des CIA 1 laden
8217   48         PHA          und retten
8218   A0 FF      LDY #$FF     Joystickeingabe setzen
821A   8C 00 DC   STY $DC00
821D   BD 00 DC   LDA $DC00,X  Joystickwert holen
8220   DD 00 DC   CMP $DC00,X  Entprellen
8223   D0 F8      BNE $821D
8225   AA         TAX          Wert in (X) retten
8226   68         PLA          Port A des CIA 1 wieder holen
8227   8D 00 DC   STA $DC00    und setzen
822A   8A         TXA          Joystickwert in (A)
822B   28         PLP          Tastaturabfrage wieder erlauben
822C   29 0F      AND #$0F     Wert korrigieren
822E   A8         TAY
822F   B9 3D 82   LDA $823D,Y  Ergebnis aus Tabelle holen
8232   A8         TAY
8233   8A         TXA
8234   29 10      AND #$10     Fire ?
8236   D0 04      BNE $823C    Nein, Skip
```

```
8238   98         TYA
8239   09 80      ORA #$80      Fire-Bit setzen
823B   A8         TAY
823C   4C D4 84   JMP $84D4     Byte-Wert in FAC#1
823F   4C 28 7D   JMP $7D28     'ILLEGAL QUANTITY'

****************************** Tabelle mit Joystickwerten

8242   04 02 03 00 06 08 07 00
824A   05 01 00

****************************** BASIC-Funktion POT

824D   20 56 79   JSR $7956     Test auf ')'
8250   20 F7 87   JSR $87F7     Byte-Wert holen
8253   CA         DEX           Wert anpassen
8254   E0 04      CPX #$04      > 4 ?
8256   B0 53      BCS $82AB     Ja, Fehler
8258   20 45 A8   JSR $A845     ROMs einschalten
825B   A0 00      LDY #$00      Paddlenummer auf 0
825D   8A         TXA           Paddlenummer in (A)
825E   A2 40      LDX #$40      Paddlesatz-Auswahl 1
8260   4A         LSR           Paddle 1 oder 3 ?
8261   90 01      BCC $8264     Nein, Skip
8263   C8         INY           Paddlenummer auf 1
8264   4A         LSR           Paddle 0 oder 1 ?
8265   90 02      BCC $8269     Ja, Skip
8267   A2 80      LDX #$80      Paddlesatz-Auswahl 2
8269   8E B1 12   STX $12B1     Auswahlwert setzen
826C   08         PHP           Status retten
826D   78         SEI           Tastaturabfrage verbieten
826E   AD 00 DC   LDA $DC00     Port A retten
8271   48         PHA
8272   8E 00 DC   STX $DC00     Paddlesatz setzen
8275   A2 00      LDX #$00      Auf A/D-Wandlung warten
8277   E8         INX
8278   D0 FD      BNE $8277
827A   B9 19 D4   LDA $D419,Y   Paddlewert lesen
827D   D9 19 D4   CMP $D419,Y   Entprellen
8280   D0 F8      BNE $827A
8282   8D B2 12   STA $12B2     Paddlewert retten
8285   A2 00      LDX #$00      Paddlesatznummer auf 0
8287   2C B1 12   BIT $12B1     Zweiter Paddlesatz ?
828A   30 01      BMI $828D     Ja, Skip
828C   E8         INX           oder falls nötig auf 1
828D   A9 04      LDA #$04      Fire-Testwert laden
828F   88         DEY           Anderes Paddle ?
8290   30 01      BMI $8293     Nein, Skip
```

```
8292   0A         ASL          Falls nötig * 2
8293   A0 FF      LDY #$FF     Port A programmieren
8295   8C 00 DC   STY $DC00
8298   C8         INY          (Y) auf 0
8299   3D 00 DC   AND $DC00,X  Fire ?
829C   D0 01      BNE $829F    Nein, Skip
829E   C8         INY          (Y) auf 1 ($01xx)
829F   68         PLA          Port A holen
82A0   8D 00 DC   STA $DC00    und wieder setzen
82A3   98         TYA          Fire-Wert in (A)
82A4   AC B2 12   LDY $12B2    Paddlewert holen
82A7   28         PLP          Tastaturabfrage wieder erlauben
82A8   4C C9 84   JMP $84C9    (Y)/(A) in FAC#1
82AB   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'

****************************** BASIC-Funktion PEN

82AE   20 56 79   JSR $7956    Test auf ')'
82B1   20 F7 87   JSR $87F7    Byte-Wert holen
82B4   E0 05      CPX #$05     > 4 ?
82B6   B0 F3      BCS $82AB    Ja, Fehler
82B8   E0 02      CPX #$02     > 1 ?
82BA   B0 1A      BCS $82D6    Ja, Skip
82BC   BC E9 11   LDY $11E9,X  Lightpenwert aus Tabelle holen
82BF   8C B1 12   STY $12B1    Wert setzen
82C2   A9 00      LDA #$00     und Tabellenwert
82C4   9D E9 11   STA $11E9,X  löschen
82C7   E0 00      CPX #$00     X-Koordinate ?
82C9   D0 05      BNE $82D0    Nein, Skip
82CB   0E B1 12   ASL $12B1    Wert = Wert * 2
82CE   69 00      ADC #$00     High-Byte setzen
82D0   AC B1 12   LDY $12B1    Lightpenwrt laden
82D3   4C C9 84   JMP $84C9    (Y)/(A) in FAC#1
82D6   20 45 A8   JSR $A845    ROMs einschalten
82D9   E0 04      CPX #$04     Aktivierungsabfrage ?
82DB   F0 10      BEQ $82ED    Ja, Skip
82DD   A0 11      LDY #$11     80-Zeichen Lightpenregister
82DF   E0 02      CPX #$02     auslesen
82E1   F0 01      BEQ $82E4
82E3   88         DEY
82E4   8C 00 D6   STY $D600
82E7   AC 01 D6   LDY $D601
82EA   4C D4 84   JMP $84D4    Byte-Wert in FAC#1
82ED   AD 00 D6   LDA $D600    Aktivierungstest
82F0   A0 00      LDY #$00     Lightpenflag auf 0
82F2   29 40      AND #$40     Lightpen aktiviert ?
82F4   F0 01      BEQ $82F7    Nein, Skip
82F6   C8         INY          Lightpenflag auf 1
```

```
82F7   4C D4 84   JMP $84D4     Byte-Wert in FAC#1

******************************* BASIC-Funktion POINTER

82FA   20 80 03   JSR $0380     CHRGOT
82FD   20 59 79   JSR $7959     Test auf '('
8300   20 3C 7B   JSR $7B3C     Buchstabe ?
8303   90 16      BCC $831B     Nein, Fehler
8305   20 AF 7A   JSR $7AAF     Variable suchen/anlegen
8308   AA         TAX           Adresse retten
8309   98         TYA
830A   48         PHA
830B   20 56 79   JSR $7956     Test auf ')'
830E   8A         TXA           Adresse wieder holen
830F   A8         TAY
8310   68         PLA
8311   C9 03      CMP #$03      High-Byte = 3 ($03D2 Interpretervariable)
8313   D0 03      BNE $8318     Nein, Skip
8315   A9 00      LDA #$00      $0000 als Adresse setzen
8317   A8         TAY           High-Byte auch auf 0
8318   4C C9 84   JMP $84C9     Integerwert in FAC#1
831B   4C 6C 79   JMP $796C     'SYNTAX'

******************************* BASIC-Funktion RSPRITE

831E   20 F7 87   JSR $87F7     Spritenummer auswerten
8321   CA         DEX           Wert anpassen
8322   E0 10      CPX #$10      Fehler ! CPX #$08 (Nur 8 Sprites !)
8324   B0 32      BCS $8358     Wenn zu groß, Fehler
8326   8A         TXA           Spritenummer retten
8327   48         PHA
8328   20 5C 79   JSR $795C     Test auf ','
832B   20 F4 87   JSR $87F4     Abfragewert auswerten
832E   20 56 79   JSR $7956     Test auf ')'
8331   E0 06      CPX #$06      Abfragewert zu groß ?
8333   B0 23      BCS $8358     Ja, Fehler
8335   68         PLA           Spritenummer laden
8336   A8         TAY           Spritenummer als Offset laden
8337   20 45 A8   JSR $A845     ROMs einschalten
833A   B9 27 D0   LDA $D027,Y   Spritefarbe laden
833D   29 0F      AND #$0F      normalisieren
833F   18         CLC           und in Bereich von 1 bis 16 bringen
8340   69 01      ADC #$01      durch Addition von 1
8342   E0 01      CPX #$01      War Spritefarbe gefragt ?
8344   F0 0E      BEQ $8354     Ja, Skip
8346   BD 5B 83   LDA $835B,X   Offset auf VIC-Register laden
8349   AA         TAX           und setzen
834A   B9 B3 6C   LDA $6CB3,Y   Bitmuster für Sprite laden
```

```
834D   3D 00 D0   AND $D000,X  und Bit testen
8350   F0 02      BEQ $8354    Wenn 0, Skip
8352   A9 01      LDA #$01     Wert 1 übergeben
8354   A8         TAY
8355   4C D4 84   JMP $84D4    Byte-Wert in FAC#1
8358   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'

****************************** Offsets auf VIC-Register für RSPRITE

835B   15 27 1B 1D 17 1C

****************************** BASIC-Funktion RSPCOLOR

8361   20 56 79   JSR $7956    Test auf '('
8364   20 F7 87   JSR $87F7    Zusatzfarbnummer auswerten
8367   CA         DEX          Wert anpassen
8368   E0 02      CPX #$02     Wert zu groß ?
836A   B0 0D      BCS $8379    Ja, Fehler
836C   20 45 A8   JSR $A845    ROMs einschalten
836F   BD 25 D0   LDA $D025,X  Zusatzfarbe laden
8372   29 0F      AND #$0F     normalisieren
8374   A8         TAY
8375   C8         INY          und in Bereich von 1 bis 16 bringen
8376   4C D4 84   JMP $84D4    Byte-Wert in FAC#1
8379   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'

****************************** BASIC-Funktion BUMP

837C   20 56 79   JSR $7956    Test auf '('
837F   20 F7 87   JSR $87F7    Kollisionswert holen
8382   CA         DEX          Wet anpassen
8383   E0 02      CPX #$02     Wert zu groß ?
8385   B0 0D      BCS $8394    Ja, Fehler
8387   78         SEI          Interrupt sperren
8388   BC E7 11   LDY $11E7,X  Spritenummer der letzten Kollision laden
838B   A9 00      LDA #$00     und Kollisionsflag in
838D   9D E7 11   STA $11E7,X  VIC-tabelle löschen
8390   58         CLI          Interrupt wieder freigeben
8391   4C D4 84   JMP $84D4    Byte-Wert in FAC#1
8394   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'

****************************** BASIC-Funktion RSPPOS

8397   20 F7 87   JSR $87F7    Spritenummer auswerten
839A   CA         DEX          Wet anpassen
839B   E0 10      CPX #$10     Fehler CPX #$08 (Nur 8 Sprites !)
839D   B0 3F      BCS $83DE    Wenn Wert zu groß, Fehler
839F   8A         TXA          Spritenummer retten
```

```
83A0  48        PHA
83A1  20 5C 79  JSR $795C    Test auf ','
83A4  20 F4 87  JSR $87F4    Abfragewert laden
83A7  20 56 79  JSR $7956    Test auf ')'
83AA  E0 03     CPX #$03     Abfragewert zu groß ?
83AC  B0 30     BCS $83DE    Ja, Fehler
83AE  68        PLA          Spritenummer wieder holen
83AF  A8        TAY          und als Offset laden
83B0  E0 02     CPX #$02     Geschwindigkeit gefragt ?
83B2  D0 09     BNE $83BD    Nein, Skip
83B4  BE D9 6D  LDX $6DD9,Y  Offset für Spritedaten laden
83B7  BC 7E 11  LDY $117E,X  und Geschwindigkeit laden
83BA  4C D4 84  JMP $84D4    Byte-Wert in FAC#1
83BD  78        SEI          Interrupt sperren
83BE  B9 B3 6C  LDA $6CB3,Y  Bitmuster für Sprite laden
83C1  2D E6 11  AND $11E6    und MSB der X-Position testen
83C4  F0 02     BEQ $83C8    Wenn 0, Skip
83C6  A9 01     LDA #$01     MSB = 1
83C8  48        PHA          High-Byte der X-Position retten
83C9  98        TYA          Offset verdoppeln
83CA  0A        ASL          * 2
83CB  A8        TAY          Offset wieder setzen
83CC  8A        TXA
83CD  4A        LSR          Y-Position holen ?
83CE  90 05     BCC $83D5    Nein, Skip
83D0  C8        INY          Offset erhöhen
83D1  68        PLA          X-Wert vom Stapel
83D2  A9 00     LDA #$00     0 als High-Byte für Y-Wert
83D4  48        PHA          auf Stapel legen
83D5  B9 D6 11  LDA $11D6,Y  Position Low-Byte laden
83D8  58        CLI          Interrupt wieder freigeben
83D9  A8        TAY          Low-Byte setzen
83DA  68        PLA          High-Byte holen
83DB  4C C9 84  JMP $84C9    Integerwert in FAC#1
83DE  4C 28 7D  JMP $7D28    'ILLEGAL QUANTITY'

****************************** BASIC-Funktion XOR

83E1  A5 16     LDA $16      Adresse Low-Byte laden
83E3  48        PHA          und retten
83E4  A5 17     LDA $17      Adresse High-Byte laden
83E6  48        PHA          und retten
83E7  20 DA 77  JSR $77DA    Auf numerisch prüfen
83EA  20 15 88  JSR $8815    FAC#1 in Adreßformat
83ED  48        PHA          Wert retten
83EE  98        TYA
83EF  48        PHA
83F0  20 0F 88  JSR $880F    Adreßwert nach Komma holen
```

```
83F3   20 56 79   JSR $7956   Test auf ')'
83F6   68         PLA         Low-Byte holen
83F7   45 16      EOR $16     Werte verknüpfen
83F9   A8         TAY         und Low-Byte in (Y)
83FA   68         PLA         High-Byte holen
83FB   45 17      EOR $17     und Werte verknüpfen
83FD   20 C9 84   JSR $84C9   Integerwert in FAC#1
8400   68         PLA         Adresse High-Byte weider holen
8401   85 17      STA $17     und setzen
8403   68         PLA         Adresse Low-Byte wieder holen
8404   85 16      STA $16     und setzen
8406   60         RTS

****************************** BASIC-Funktion RWINDOW

8407   20 56 79   JSR $7956   Test auf '('
840A   20 F7 87   JSR $87F7   Abfragewert holen
840D   E0 02      CPX #$02    Bildschirmbreite gefragt ?
840F   F0 14      BEQ $8425   Ja, Skip
8411   B0 1E      BCS $8431   Wenn falscher Wert, Fehler
8413   E0 00      CPX #$00    Zeilenbreite gefragt ?
8415   D0 07      BNE $841E   Nein, Skip
8417   A5 E4      LDA $E4     Zeilenbreite des Fensters ausrechnen
8419   38         SEC
841A   E5 E5      SBC $E5
841C   B0 0F      BCS $842D   Unbedingter Sprung
841E   A5 E7      LDA $E7     Spaltenbreite des Fensters ausrechnen
8420   38         SEC
8421   E5 E6      SBC $E6
8423   B0 08      BCS $842D   Unbedingter Sprung
8425   A9 28      LDA #$28    Schirmbreite 40 Zeichen
8427   24 D7      BIT $D7     80-Zeichen-Modus ?
8429   10 02      BPL $842D   Nein, Skip
842B   A9 50      LDA #$50    Schirmbreite 80-Zeichen
842D   A8         TAY
842E   4C D4 84   JMP $84D4   Byte-Wert in FAC#1
8431   4C 28 7D   JMP $7D28   'ILLEGAL QUANTITY'

****************************** BASIC-Funktion RND

8434   20 57 8C   JSR $8C57   Vorzeichen holen
8437   30 31      BMI $846A   Wenn negativ, Skip
8439   D0 1A      BNE $8455   Wenn positiv, Skip
843B   20 45 A8   JSR $A845   ROMs einschalten
843E   AD 06 DC   LDA $DC06   Timer B in FAC#1
8441   85 64      STA $64
8443   AD 07 DC   LDA $DC07
8446   85 66      STA $66
```

```
8448   AD 04 DC   LDA $DC04    Timer A in FAC#1
844B   85 65      STA $65
844D   AD 05 DC   LDA $DC05
8450   85 67      STA $67
8452   4C 7A 84   JMP $847A
8455   A9 1B      LDA #$1B     Zeiger auf letzten RND-Wert
8457   A0 12      LDY #$12     ab $121B laden
8459   20 D4 8B   JSR $8BD4    Wert in FAC#1
845C   A9 90      LDA #$90     Zeiger auf Konstante 1
845E   A0 84      LDY #$84     ab $8490 laden
8460   20 08 8A   JSR $8A08    FAC#1 = FAC#1 * Konstante
8463   A9 95      LDA #$95     Zeiger auf Konstante 2
8465   A0 84      LDY #$84     ab $8495 laden
8467   20 12 8A   JSR $8A12    FAC#1 = FAC#1 + Konstante
846A   A6 67      LDX $67      Stellen des FAC#1 vertauschen
846C   A5 64      LDA $64
846E   85 67      STA $67
8470   86 64      STX $64
8472   A6 65      LDX $65
8474   A5 66      LDA $66
8476   85 65      STA $65
8478   86 66      STX $66
847A   A9 00      LDA #$00     Vorzeichen löschen
847C   85 68      STA $68
847E   A5 63      LDA $63      Exponent in
8480   85 71      STA $71      Rundungsbyte setzen
8482   A9 80      LDA #$80     Exponent uf Bereich 0-1
8484   85 63      STA $63      setzen
8486   20 B6 88   JSR $88B6    FAC#1 linksbündig machen
8489   A2 1B      LDX #$1B     Zeiger auf letzten RND-Wert
848B   A0 12      LDY #$12     ab $121B setzen
848D   4C 00 8C   JMP $8C00    FAC#1 speichern

******************************** Konstanten für RND

8490   98 35 44 7A 00          11879546
8495   68 28 B1 46 00          3.92767774E-4

******************************** Konstante - 32768

849A   90 80 00 00 00          -32768

******************************** Umwandlung FAC#1 zu Integer

849F   20 B4 84   JSR $84B4    FAC#1 in Integerformat
84A2   A5 66      LDA $66      Wert in (A)/(Y) laden
84A4   A4 67      LDY $67
```

```
84A6   60          RTS

****************************** Ausdruck holen und in Integer

84A7   20 80 03    JSR $0380     CHRGET
84AA   20 EF 77    JSR $77EF     FRMEVL Ausdruck auswerten
84AD   20 DA 77    JSR $77DA     Auf numerisch prüfen
84B0   A5 68       LDA $68       Wert negativ ?
84B2   30 0D       BMI $84C1     Ja, Fehler
84B4   A5 63       LDA $63       Exponent laden
84B6   C9 90       CMP #$90      Wert ungültig ?
84B8   90 0C       BCC $84C6     Nein, Skip
84BA   A9 9A       LDA #$9A      Zeiger auf Konstante -32768
84BC   A0 84       LDY #$84      ab $849A laden
84BE   20 87 8C    JSR $8C87     FAC#1 = Konstante ?
84C1   F0 03       BEQ $84C6     Ja, Skip
84C3   4C 28 7D    JMP $7D28     'ILLEGAL QUANTITY'
84C6   4C C7 8C    JMP $8CC7     FAC#1 in Integerformat

****************************** Integerwert in FAC#1

84C9   20 E5 84    JSR $84E5     Vorbereitung für Integer (Y)/(A) in FAC#1
84CC   38          SEC           Flag für Adreßwert setzen
84CD   4C 75 8C    JMP $8C75     Adreßwert in FAC#1

****************************** BASIC-Funktion POS

84D0   38          SEC           Flag für Position lesen
84D1   20 8D 92    JSR $928D     CURSOR-Position holen
84D4   A9 00       LDA #$00      (A)/(X) in FAC#1
84D6   4C 3C 79    JMP $793C     Wert in FAC#1

****************************** Test auf Direktmodus

84D9   24 7F       BIT $7F       Direktmodus ?
84DB   30 12       BMI $84EF     Nein, Skip
84DD   A2 15       LDX #$15      'ILLEGAL DIRECT'
84DE   2C          .BYTE $2C     Nächsten Befehl überlesen
84DF   A2 1B       LDX #$1B      'UNDEFINED FUNCTION'
84E2   4C 3C 4D    JMP $4D3C     Fehler ausgeben

****************************** Vorbereitung für Umwandlung
                               in Fließkommaformat

84E5   A2 00       LDX #$00      Flag auf numerisch
84E7   86 0F       STX $0F       setzen
84E9   85 64       STA $64       Wert High-Byte (!) setzen
84EB   84 65       STY $65       Wert Low-Byte (!) setzen
```

```
84ED   A2 90      LDX #$90    Exponent laden
84EF   60         RTS

****************************** Test auf Programm-Modus

84F0   24 7F      BIT $7F     Direktmodus ?
84F2   30 01      BMI $84F5   Nein, Fehler
84F4   60         RTS
84F5   A2 22      LDX #$22    'DIRECT MODE ONLY'
84F7   4C 3C 4D   JMP $4D3C   Fehler ausgeben

****************************** BASIC-Befehl DEF FN

84FA   20 28 85   JSR $8528   FN-Syntax prüfen
84FD   20 D9 84   JSR $84D9   Direktmodus ?
8500   20 59 79   JSR $7959   Test auf '('
8503   A9 80      LDA #$80    Integerwerte sperren
8505   85 12      STA $12
8507   20 AF 7A   JSR $7AAF   Variable suchen
850A   20 DA 77   JSR $77DA   Auf numerisch prüfen
850D   20 56 79   JSR $7956   Auf ')' prüfen
8510   A9 B2      LDA #$B2    Token für '='
8512   20 5E 79   JSR $795E   Prüft auf Code
8515   48         PHA         Aktuelles Zeichen retten
8516   A5 4A      LDA $4A     FN-Variablenadresse High-Byte
8518   48         PHA         auf Stapel
8519   A5 49      LDA $49     FN-Variablenadresse Low-Byte
851B   48         PHA         auf Stapel
851C   A5 3E      LDA $3E     PC High-Byte auf Stapel
851E   48         PHA         retten
851F   A5 3D      LDA $3D     PC Low-Byte auf Stapel
8521   48         PHA         retten
8522   20 8F 52   JSR $528F   DATA-Befehl
8525   4C A0 85   JMP $85A0   FN-Variable vom Stack holen

****************************** Auf FN-Syntax prüfen

8528   A9 A5      LDA #$A5    Token für FN
852A   20 5E 79   JSR $795E   Prüft auf Code
852D   09 80      ORA #$80    Integerwerte sperren
852F   85 12      STA $12
8531   20 B6 7A   JSR $7AB6   Variable suchen
8534   85 50      STA $50     Adresse Low-Byte speichern
8536   84 51      STY $51     Adresse High-Byte speichern
8538   4C DA 77   JMP $77DA   Auf numerisch prüfen
```

```
******************************** BASIC-Funktion FN

853B   20 28 85   JSR $8528   FN-Syntax prüfen
853E   A5 51      LDA $51     FN-Variablenzeiger High-Byte
8540   48         PHA         auf Stapel retten
8541   A5 50      LDA $50     FN-Variablenzeiger Low-Byte
8543   48         PHA         auf Stapel retten
8544   20 50 79   JSR $7950   Ausdruck in Klammern auswerten
8547   20 DA 77   JSR $77DA   Auf numerisch prüfen
854A   68         PLA         Adresse wieder holen
854B   85 50      STA $50     Adresse Low-Byte setzen
854D   68         PLA         Adresse High-Byte wieder holen
854E   85 51      STA $51     und setzen
8550   A0 02      LDY #$02    Offset laden
8552   20 CE 42   JSR $42CE   Adresse Low-Byte aus Variable holen
8555   85 49      STA $49     und setzen
8557   AA         TAX         Wert auch in (X) retten
8558   C8         INY         Offset erhöhen
8559   20 CE 42   JSR $42CE   Adresse High-Byte aus Variable holen
855C   F0 82      BEQ $84E0   Wenn 0, 'UNDEF'D FUNCTION'
855E   85 4A      STA $4A     Adresse High-Byte setzen
8560   C8         INY         Offset erhöhen
8561   A9 49      LDA #$49    ergibt LDA ($49),Y
8563   20 AB 03   JSR $03AB   Byte aus FN-variable holen
8566   48         PHA         und retten
8567   88         DEY         Alle Bytes der Variablen gerettet ?
8568   10 F7      BPL $8561   Nein, weitermachen
856A   A4 4A      LDY $4A     Adresse High-Byte laden
856C   8D 04 FF   STA $FF04   Write in Bank 2 setzen
856F   20 00 8C   JSR $8C00   FAC#1 in FN-Variable
8572   A5 3E      LDA $3E     PC High-Byte laden
8574   48         PHA         und retten
8575   A5 3D      LDA $3D     PC Low-Byte laden
8577   48         PHA         und retten
8578   20 CE 42   JSR $42CE   FN-PC setzen
857B   85 3D      STA $3D     PC Low-Byte setzen
857D   C8         INY         Offset erhöhen
857E   20 CE 42   JSR $42CE   PC High-Byte laden
8581   85 3E      STA $3E     und setzen
8583   A5 4A      LDA $4A     Variablenadresse High-Byte laden
8585   48         PHA         und retten
8586   A5 49      LDA $49     Variablenadresse Low-Byte laden
8588   48         PHA         und retten
8589   20 D7 77   JSR $77D7   FRMNUM Numerischen Ausdruck auswerten
858C   68         PLA         Variablenadresse Low-Byte holen
858D   85 50      STA $50     Adresse neu setzen
858F   68         PLA         Variablenadresse High-Byte holen
8590   85 51      STA $51     Adresse neu setzen
```

```
8592   20 86 03   JSR $0386     CHRGOT
8595   F0 03      BEQ $859A     Wenn Trennzeichen, Skip
8597   4C 6C 79   JMP $796C     'SYNTAX'
859A   68         PLA           PC Low-Byte wieder holen
859B   85 3D      STA $3D       und setzen
859D   68         PLA           PC High-Byte wieder holen
859E   85 3E      STA $3E       und setzen
85A0   A0 00      LDY #$00      Offset auf 0
85A2   8D 04 FF   STA $FF04     Write in Bank 1 setzen
85A5   68         PLA           Variablenwert wieder holen
85A6   91 50      STA ($50),Y   Variablenbytes eintragen
85A8   C8         INY           Offset erhöhen
85A9   C0 05      CPY #$05      FN-Variable wieder gesetzt ?
85AB   D0 F8      BNE $85A5     Nein, weitermachen
85AD   60         RTS

******************************* BASIC-Funktion STR$

85AE   20 DA 77   JSR $77DA     Auf numerisch prüfen
85B1   A0 00      LDY #$00      Offset auf 0
85B3   20 44 8E   JSR $8E44     FAC#1 in ASCII-String umwandeln
85B6   68         PLA           Rücksprungadresse vom
85B7   68         PLA           Stapel löschen
85B8   A9 FF      LDA #$FF      Stringstartadresse = $00FF
85BA   A0 00      LDY #$00
85BC   4C 9A 86   JMP $869A     String in Stringbereich kopieren

******************************* BASIC-Funktion CHR$

85BF   20 F7 87   JSR $87F7     Holt Byte-Wert
85C2   8A         TXA           Wert retten
85C3   48         PHA
85C4   A9 01      LDA #$01      Platz für 1 Zeichen im Stringspeicher
85C6   20 90 86   JSR $8690     reservieren
85C9   68         PLA           Wert holen
85CA   A0 00      LDY #$00      Offset auf 0 setzen
85CC   8D 04 FF   STA $FF04     Write in Bank 1 setzen
85CF   91 64      STA ($64),Y   Zeichen in Stringspeicher
85D1   68         PLA           Rücksprungadresse vom
85D2   68         PLA           Stapel löschen
85D3   4C E3 86   JMP $86E3     String in Stringstack

******************************* BASIC-Funktion LEFT$

85D6   20 4D 86   JSR $864D     Stringparameter holen
85D9   48         PHA           Länge retten
85DA   20 D8 42   JSR $42D8     LEFT$-Parameter holen
85DD   85 79      STA $79       parameter setzen
```

```
85DF   68         PLA         Länge wieder holen
85E0   C5 79      CMP $79     LEFT$-Parameter < Stringlänge ?
85E2   98         TYA         (A) = 0
85E3   90 05      BCC $85EA   Ja, Skip
85E5   20 D8 42   JSR $42D8
85E8   AA         TAX
85E9   98         TYA
85EA   48         PHA
85EB   8A         TXA
85EC   48         PHA
85ED   20 90 86   JSR $8690   Platz für String reservieren
85F0   A5 52      LDA $52     Zeiger auf Deskriptor
85F2   A4 53      LDY $53
85F4   20 85 87   JSR $8785   FRESTR
85F7   68         PLA
85F8   A8         TAY
85F9   68         PLA         Länge des neuen Strings
85FA   18         CLC
85FB   65 24      ADC $24     + Adresse des alten Strings
85FD   85 24      STA $24
85FF   90 02      BCC $8603   Kein Übertrag, Skip
8601   E6 25      INC $25     Übertrag setzen
8603   98         TYA
8604   20 63 87   JSR $8763   Neuen String in Stringbereich schreiben
8607   4C E3 86   JMP $86E3   Deskriptor in Stringstack setzen

******************************** BASIC-Befehl RIGHT$

860A   20 4D 86   JSR $864D   Stringparameter holen
860D   48         PHA         Länge retten
860E   20 D8 42   JSR $42D8   RIGHT$-Parameter setzen
8611   85 79      STA $79
8613   68         PLA         Länge holen
8614   18         CLC
8615   E5 79      SBC $79     - Länge des Parameters
8617   49 FF      EOR #$FF    = 1. Position im alten String
8619   4C E3 85   JMP $85E3   Zu LEFT$

******************************** BASIC-Funktion MID$

861C   A9 FF      LDA #$FF    Länge des Ausschnitts auf
861E   85 67      STA $67     Maximalwert
8620   20 86 03   JSR $0386   CHRGET
8623   C9 29      CMP #$29    ')' ?
8625   F0 06      BEQ $862D   Ja, Skip
8627   20 5C 79   JSR $795C   Test auf ','
862A   20 F4 87   JSR $87F4   Byte-Wert holen
862D   20 4D 86   JSR $864D   Stringparameter holen
```

```
8630   F0 53      BEQ $8685    Wenn Länge = 0, Fehler
8632   CA         DEX
8633   8A         TXA
8634   48         PHA
8635   A2 00      LDX #$00
8637   48         PHA
8638   20 D8 42   JSR $42D8    Länge des alten Strings setzen
863B   85 79      STA $79
863D   68         PLA
863E   18         CLC
863F   E5 79      SBC $79      1. Position < Stringlänge
8641   B0 A8      BCS $85EB    Ja, zu LEFT$
8643   49 FF      EOR #$FF     Wert anpassen
8645   C5 67      CMP $67      < Ausschnittlänge ?
8647   90 A3      BCC $85EC    Ja, zu LEFT$
8649   A5 67      LDA $67      mit Ausschnittlänge zu LEFT$
864B   B0 9F      BCS $85EC    Unbedingter Sprung

****************************** Stringparameter vom Stack holen

864D   20 56 79   JSR $7956    Test auf ')'
8650   68         PLA          Rücksprungadresse retten
8651   A8         TAY
8652   68         PLA
8653   85 57      STA $57
8655   68         PLA          Stringadresse vom
8656   68         PLA          Stapel löschen
8657   68         PLA          1. Parameter vom Stack holen
8658   AA         TAX          Länge merken
8659   68         PLA          Adresse vom Stack holen
865A   85 52      STA $52
865C   68         PLA
865D   85 53      STA $53
865F   A5 57      LDA $57      Rücksprungadresse wieder auf den Stack
8661   48         PHA
8662   98         TYA
8663   48         PHA
8664   A0 00      LDY #$00     Offset (Y) auf 0
8666   8A         TXA          Länge in (A)
8667   60         RTS

****************************** BASIC-Funktion LEN

8668   20 6E 86   JSR $866E    FRESTR und Länge holen
866B   4C D4 84   JMP $84D4    Byte-Wert in FAC#1
```

```
834D   3D 00 D0   AND $D000,X  und Bit testen
8350   F0 02      BEQ $8354    Wenn 0, Skip
8352   A9 01      LDA #$01     Wert 1 übergeben
8354   A8         TAY
8355   4C D4 84   JMP $84D4    Byte-Wert in FAC#1
8358   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'

****************************** Offsets auf VIC-Register für RSPRITE

835B   15 27 1B 1D 17 1C

****************************** BASIC-Funktion RSPCOLOR

8361   20 56 79   JSR $7956    Test auf '('
8364   20 F7 87   JSR $87F7    Zusatzfarbnummer auswerten
8367   CA         DEX          Wert anpassen
8368   E0 02      CPX #$02     Wert zu groß ?
836A   B0 0D      BCS $8379    Ja, Fehler
836C   20 45 A8   JSR $A845    ROMs einschalten
836F   BD 25 D0   LDA $D025,X  Zusatzfarbe laden
8372   29 0F      AND #$0F     normalisieren
8374   A8         TAY
8375   C8         INY          und in Bereich von 1 bis 16 bringen
8376   4C D4 84   JMP $84D4    Byte-Wert in FAC#1
8379   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'

****************************** BASIC-Funktion BUMP

837C   20 56 79   JSR $7956    Test auf '('
837F   20 F7 87   JSR $87F7    Kollisionswert holen
8382   CA         DEX          Wet anpassen
8383   E0 02      CPX #$02     Wert zu groß ?
8385   B0 0D      BCS $8394    Ja, Fehler
8387   78         SEI          Interrupt sperren
8388   BC E7 11   LDY $11E7,X  Spritenummer der letzten Kollision laden
838B   A9 00      LDA #$00     und Kollisionsflag in
838D   9D E7 11   STA $11E7,X  VIC-tabelle löschen
8390   58         CLI          Interrupt wieder freigeben
8391   4C D4 84   JMP $84D4    Byte-Wert in FAC#1
8394   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'

****************************** BASIC-Funktion RSPPOS

8397   20 F7 87   JSR $87F7    Spritenummer auswerten
839A   CA         DEX          Wet anpassen
839B   E0 10      CPX #$10     Fehler CPX #$08 (Nur 8 Sprites !)
839D   B0 3F      BCS $83DE    Wenn Wert zu groß, Fehler
839F   8A         TXA          Spritenummer retten
```

```
86B2  F0 04     BEQ $86B8     Ja, Skip
86B4  C5 0A     CMP $0A       Endezeichen 2 ?
86B6  D0 F2     BNE $86AA     Nein, weitersuchen
86B8  C9 22     CMP #$22      '"' ?
86BA  F0 01     BEQ $86BD     Ja, Skip
86BC  18        CLC
86BD  84 63     STY $63       Länge des Strings setzen
86BF  98        TYA           Offset als Länge laden
86C0  65 70     ADC $70       Länge + Startadresse
86C2  85 72     STA $72       = Endadresse +1
86C4  A6 71     LDX $71
86C6  90 01     BCC $86C9     Kein Übertrag, Skip
86C8  E8        INX           Übertrag berücksichtigen
86C9  86 73     STX $73       und High-Byte setzen
86CB  98        TYA

86CC  20 88 86  JSR $8688     Stringzeiger berechnen
86CF  A8        TAY           Länge = 0 ?
86D0  F0 11     BEQ $86E3     Ja, Skip
86D2  48        PHA           Länge retten
86D3  88        DEY           Länge als Zähler erniedrigen
86D4  20 F1 42  JSR $42F1     Zeichen des Strings holen
86D7  8D 04 FF  STA $FF04     Write in Bank 1 setzen
86DA  91 37     STA ($37),Y   und umkopieren
86DC  98        TYA           weitermachen
86DD  D0 F4     BNE $86D3     bis alle Zeichen umkopiert
86DF  68        PLA           Länge holen
86E0  20 71 87  JSR $8771     Hilfszeiger erhöhen

***************************** Stringzeiger in Deskriptorstack

86E3  A6 18     LDX $18       Stringstackpointer
86E5  E0 24     CPX #$24      Stringstack voll ?
86E7  D0 05     BNE $86EE     Nein, Skip
86E9  A2 19     LDX #$19      'FORMULA TOO COMPLEX'
86EB  4C 3C 4D  JMP $4D3C     Fehler ausgeben
86EE  A5 63     LDA $63       Stringlänge laden
86F0  95 00     STA $00,X     und in Stringstack setzen
86F2  A5 64     LDA $64       Adresse Low-Byte laden
86F4  95 01     STA $01,X     und in Stringstack setzen
86F6  A5 65     LDA $65       Adresse High-Byte laden
86F8  95 02     STA $02,X     und in Stringstack setzen
86FA  A0 00     LDY #$00      0 als High-Byte laden
86FC  86 66     STX $66       Zeiger auf Deskriptor setzen
86FE  84 67     STY $67       High-Byte auf 0 setzen
8700  84 71     STY $71       Rundungsbyte FAC#1 löschen
8702  88        DEY           (Y) = $FF
8703  84 0F     STY $0F       Stringflag setzen
```

```
8705    86 19       STX $19      Index des letzten Stringdeskriptors
8707    E8          INX          Zeiger auf Stringstack
8708    E8          INX          um 3 erhöhen
8709    E8          INX
870A    86 18       STX $18      = Neuer Deskriptorstackpointer
870C    60          RTS

****************************** Stringverknüpfung '+'

870D    A5 67       LDA $67      Adresse des ersten Srings retten
870F    48          PHA
8710    A5 66       LDA $66
8712    48          PHA
8713    20 D7 78    JSR $78D7    Zweiten String auswerten
8716    20 DD 77    JSR $77DD    Test auf String
8719    68          PLA          Adresse Low-Byte wieder holen
871A    85 70       STA $70      und setzen
871C    68          PLA          Adresse High-Byte wieder holen
871D    85 71       STA $71      und setzen
871F    A0 00       LDY #$00     Offset auf 0
8721    20 F6 42    JSR $42F6    Länge des ersten Strings holen
8724    85 79       STA $79      und speichern
8726    20 E7 42    JSR $42E7    Länge des zweiten Strings laden
8729    18          CLC
872A    65 79       ADC $79      + Länge des ersten Strings
872C    90 03       BCC $8731    Wenn < 256, Skip
872E    4C ED A5    JMP $A5ED    'STRING TOO LONG'
8731    20 88 86    JSR $8688    Platz für String reservieren
8734    20 4E 87    JSR $874E    ersten String übertragen
8737    A5 52       LDA $52      Zeiger auf zweiten String
8739    A4 53       LDY $53      laden
873B    20 85 87    JSR $8785    FRESTR
873E    20 63 87    JSR $8763    zweiten String danach einsetzen
8741    A5 70       LDA $70      Zeiger auf ersten String
8743    A4 71       LDY $71      laden
8745    20 85 87    JSR $8785    FRESTR
8748    20 E3 86    JSR $86E3    Deskriptor in Stringstack
874B    4C 09 78    JMP $7809    Zurück zu FRMEVL

****************************** String in reservierten Bereich übertragen

874E    A0 00       LDY #$00     Offset auf 0 setzen
8750    20 F6 42    JSR $42F6    Länge laden
8753    48          PHA          und retten
8754    C8          INY          Offset erhöhen
8755    20 F6 42    JSR $42F6    Adresse Low-Byte laden
8758    AA          TAX          und in (X) merken
8759    C8          INY          Offset erhöhen
```

```
875A   20 F6 42   JSR $42F6      Adresse High-Byte laden
875D   A8         TAY            und in (Y) retten
875E   68         PLA            Länge holen
875F   86 24      STX $24        und Adresse Low-Byte setzen
8761   84 25      STY $25        und Adresse High-Byte setzen

****************************** String übertragen

8763   A8         TAY            Länge = 0 ?
8764   F0 0B      BEQ $8771      Ja, Skip
8766   48         PHA            Länge retten
8767   88         DEY            Länge als Zähler erniedrigen
8768   20 B7 03   JSR $03B7      Zeichen holen
876B   91 37      STA ($37),Y    und in Stringbereich übertragen
876D   98         TYA            Alle Zeichen übertragen ?
876E   D0 F7      BNE $8767      Nein, weitermachen
8770   68         PLA            Länge wieder laden

****************************** Hilfszeiger erhöhen

8771   18         CLC
8772   65 37      ADC $37        (A) auf Hilfszeiger aufaddieren
8774   85 37      STA $37        und Low-Byte neu setzen
8776   90 02      BCC $877A      Kein Übertrag, Skip
8778   E6 38      INC $38        Übertrag berücksichtigen
877A   60         RTS

****************************** FRMEVL String + FRESTR

877B   20 EF 77   JSR $77EF      FRMEVL Ausdruck auswerten
877E   20 DD 77   JSR $77DD      und auf String prüfen
8781   A5 66      LDA $66        Deskriptoradresse Low-Byte laden
8783   A4 67      LDY $67        Deskriptoradresse High-Byte laden

****************************** FRESTR

8785   85 24      STA $24        Deskriptoradresse Low-Byte setzen
8787   84 25      STY $25        Deskriptoradresse High-Byte setzen
8789   20 E0 87   JSR $87E0      Deskriptor vom Stringstack holen
878C   D0 3C      BNE $87CA      Wenn nicht auf Stringstack, Skip
878E   20 F6 54   JSR $54F6      Deskriptor im Stringbereich ?
8791   90 37      BCC $87CA      Nein, Adresse setzen
8793   88         DEY            Länge setzen
8794   A9 FF      LDA #$FF       Flag für Trailer ungültig laden
8796   8D 04 FF   STA $FF04      Write in Bank 1 setzen
8799   91 24      STA ($24),Y    Trailer für ungültig erklären
879B   88         DEY            Offset erniedrigen
879C   8A         TXA            Länge des Strings
```

```
879D   91 24     STA ($24),Y  als erstes Trailerbyte setzen
879F   48        PHA          und Länge retten
87A0   49 FF     EOR #$FF     Länge für Zweierkomplement invertieren
87A2   38        SEC          + 1 für Zweierkomplement (!)
87A3   65 24     ADC $24      Negative Länge auf Trailerpointer
87A5   A4 25     LDY $25      aufaddieren
87A7   B0 01     BCS $87AA    Kein Übertrag, Skip
87A9   88        DEY          High-Byte anpassen
87AA   85 24     STA $24      und neuen Zeiger auf Stringanfang
87AC   84 25     STY $25      speichern
87AE   AA        TAY          Low-Byte in (Y)
87AF   68        PLA          Stringlänge wieder holen
87B0   C4 36     CPY $36      Adresse = Anfang Stringspeicher ?
87B2   D0 3C     BNE $87F0    Nein, Ende
87B4   E4 35     CPX $35      Adresse = Anfang Stringspeicher ?
87B6   D0 38     BNE $87F0    Nein, Ende
87B8   48        PHA          Stringlänge wieder retten
87B9   38        SEC          Stringanfangszeiger korrigieren
87BA   65 35     ADC $35      indem Stringlänge + 1 (SEC!)
87BC   85 35     STA $35      aufaddiert wird
87BE   90 02     BCC $87C2    Kein Übertrag, Skip
87C0   E6 36     INC $36      Übertrag berücksichtigen
87C2   E6 35     INC $35      Anfangszeiger noch einmal erhöhen
87C4   D0 02     BNE $87C8    Kein Übertrag, Skip
87C6   E6 36     INC $36      Übertrag berücksichtigen
87C8   68        PLA          Stringlänge wieder holen
87C9   60        RTS

****************************** Deskriptorwerte in ($24) setzen

87CA   A0 00     LDY #$00     Offset auf 0 setzen
87CC   20 B7 03  JSR $03B7    Länge holen
87CF   48        PHA          und retten
87D0   C8        INY          Offset erhöhen
87D1   20 B7 03  JSR $03B7    Adresse Low-Byte holen
87D4   AA        TAX          und in (X) retten
87D5   C8        INY          Offset erhöhen
87D6   20 B7 03  JSR $03B7    Adresse High-Byte holen
87D9   A8        TAY          und in (Y) bringen
87DA   86 24     STX $24      Adresse Low-Byte setzen
87DC   84 25     STY $25      Adresse High-Byte setzen
87DE   68        PLA          Länge wieder holen
87DF   60        RTS

****************************** Stringzeiger vom Stringstack holen

87E0   C4 1A     CPY $1A      Zeigt (A)/(Y) auf Stringstack ?
87E2   D0 0C     BNE $87F0    Nein, Ende
```

```
87E4   C5 19      CMP $19      Zeigt (A)/(Y) auf Stringstack ?
87E6   D0 08      BNE $87F0    Nein, Ende
87E8   85 18      STA $18      Low-Byte als Zeiger setzen
87EA   E9 03      SBC #$03     um 3 erniedrigen (Eintrag löschen)
87EC   85 19      STA $19      und neuen Stringstackpointer setzen
87EE   A0 00      LDY #$00     Offset auf 0
87F0   60         RTS

****************************** Byte-Wert holen

87F1   20 80 03   JSR $0380    CHRGET
87F4   20 D7 77   JSR $77D7    FRMEVL Numerischen Ausdruck holen
87F7   20 AD 84   JSR $84AD    Bereich prüfen und in Integer wandeln
87FA   A6 66      LDX $66      High-Byte <> 0 ?
87FC   D0 2D      BNE $882B    Ja, Fehler
87FE   A6 67      LDX $67      Low-Byte laden
8800   4C 86 03   JMP $0386    CHRGOT

****************************** GETADR/GETBYT Adresse und Byte-Wert holen

8803   20 D7 77   JSR $77D7    FRMEVL Numerischen Ausdruck holen
8806   20 15 88   JSR $8815    FAC#1 in Adreßformat wandeln
8809   20 5C 79   JSR $795C    Test auf ','
880C   4C F4 87   JMP $87F4    Byte-Wert holen

****************************** Ausdruck nach Komma holen

880F   20 5C 79   JSR $795C    Test auf ','
8812   20 D7 77   JSR $77D7    FRMEVL Numerischen Ausdruck holen

****************************** FAC#1 in Adreßformat wandeln ($16)

8815   A5 68      LDA $68      Negativ ?
8817   30 12      BMI $882B    Ja, Fehler
8819   A5 63      LDA $63      Exponent FAC#1 laden
881B   C9 91      CMP #$91     Zahl > 65535 ?
881D   B0 0C      BCS $882B    Ja, Fehler
881F   20 C7 8C   JSR $8CC7    FAC#1 in Adreßformat wandeln
8822   A5 66      LDA $66      Adresse High-Byte (!) laden
8824   A4 67      LDY $67      Adresse Low-Byte (!) laden
8826   84 16      STY $16      Ergebnis Low-Byte setzen
8828   85 17      STA $17      Ergebnis High-Byte setzen
882A   60         RTS
882B   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'

****************************** FAC#1 = Konstante (A)/(Y) - FAC#1
```

```
****************************** BASIC-Funktion FN

853B   20 28 85   JSR $8528    FN-Syntax prüfen
853E   A5 51      LDA $51      FN-Variablenzeiger High-Byte
8540   48         PHA          auf Stapel retten
8541   A5 50      LDA $50      FN-Variablenzeiger Low-Byte
8543   48         PHA          auf Stapel retten
8544   20 50 79   JSR $7950    Ausdruck in Klammern auswerten
8547   20 DA 77   JSR $77DA    Auf numerisch prüfen
854A   68         PLA          Adresse wieder holen
854B   85 50      STA $50      Adresse Low-Byte setzen
854D   68         PLA          Adresse High-Byte wieder holen
854E   85 51      STA $51      und setzen
8550   A0 02      LDY #$02     Offset laden
8552   20 CE 42   JSR $42CE    Adresse Low-Byte aus Variable holen
8555   85 49      STA $49      und setzen
8557   AA         TAX          Wert auch in (X) retten
8558   C8         INY          Offset erhöhen
8559   20 CE 42   JSR $42CE    Adresse High-Byte aus Variable holen
855C   F0 82      BEQ $84E0    Wenn 0, 'UNDEF'D FUNCTION'
855E   85 4A      STA $4A      Adresse High-Byte setzen
8560   C8         INY          Offset erhöhen
8561   A9 49      LDA #$49     ergibt LDA ($49),Y
8563   20 AB 03   JSR $03AB    Byte aus FN-variable holen
8566   48         PHA          und retten
8567   88         DEY          Alle Bytes der Variablen gerettet ?
8568   10 F7      BPL $8561    Nein, weitermachen
856A   A4 4A      LDY $4A      Adresse High-Byte laden
856C   8D 04 FF   STA $FF04    Write in Bank 2 setzen
856F   20 00 8C   JSR $8C00    FAC#1 in FN-Variable
8572   A5 3E      LDA $3E      PC High-Byte laden
8574   48         PHA          und retten
8575   A5 3D      LDA $3D      PC Low-Byte laden
8577   48         PHA          und retten
8578   20 CE 42   JSR $42CE    FN-PC setzen
857B   85 3D      STA $3D      PC Low-Byte setzen
857D   C8         INY          Offset erhöhen
857E   20 CE 42   JSR $42CE    PC High-Byte laden
8581   85 3E      STA $3E      und setzen
8583   A5 4A      LDA $4A      Variablenadresse High-Byte laden
8585   48         PHA          und retten
8586   A5 49      LDA $49      Variablenadresse Low-Byte laden
8588   48         PHA          und retten
8589   20 D7 77   JSR $77D7    FRMNUM Numerischen Ausdruck auswerten
858C   68         PLA          Variablenadresse Low-Byte holen
858D   85 50      STA $50      Adresse neu setzen
858F   68         PLA          Variablenadresse High-Byte holen
8590   85 51      STA $51      Adresse neu setzen
```

```
8872   A0 00      LDY #$00     Rundungsbyte löschen
8874   84 71      STY $71
8876   C9 F9      CMP #$F9     Exponent zu groß ?
8878   30 C6      BMI $8840    Ja, Exponenten angleichen
887A   A8         TAY          Wert als Zähler laden
887B   A5 71      LDA $71      Rundungsbyte laden
887D   56 01      LSR $01,X    und Mantissenbits verschieben
887F   20 90 89   JSR $8990
8882   24 70       BIT $70      Haben FAC#1 und FAC#2 gleiche Vorzeichen
?
8884   10 57      BPL $88DD    Ja, Skip
8886   A0 63      LDY #$63     Offset auf FAC#1 setzen
8888   E0 6A      CPX #$6A     Ist zweiter Offset auf FAC#2 gerichtet ?
888A   F0 02      BEQ $888E    Ja, Skip
888C   A0 6A      LDY #$6A     Offset auf FAC#2 setzen
888E   38         SEC          Wert des Exponenten in (A)
888F   49 FF      EOR #$FF     negieren
8891   65 58      ADC $58      Vorzeichenvergleich aufaddieren
8893   85 71      STA $71      und Wert in Rundungsbyte setzen
8895   B9 04 00   LDA $0004,Y  Mantissen der beiden FACs
8898   F5 04      SBC $04,X    voneinander subtrahieren
889A   85 67      STA $67
889C   B9 03 00   LDA $0003,Y
889F   F5 03      SBC $03,X
88A1   85 66      STA $66
88A3   B9 02 00   LDA $0002,Y
88A6   F5 02      SBC $02,X
88A8   85 65      STA $65
88AA   B9 01 00   LDA $0001,Y
88AD   F5 01      SBC $01,X
88AF   85 64      STA $64
88B1   B0 03      BCS $88B6    Wenn kein Übertrag, Skip
88B3   20 26 89   JSR $8926    Mantisse von FAC#1 invertieren
88B6   A0 00      LDY #$00
88B8   98         TYA          Zähler auf 0
88B9   18         CLC
88BA   A6 64       LDX $64      Höchstwertiges Byte der Mantisse gesetzt
?
88BC   D0 4A      BNE $8908    Ja, Skip
88BE   A6 65      LDX $65      Mantissenbytes aufrücken
88C0   86 64      STX $64
88C2   A6 66      LDX $66
88C4   86 65      STX $65
88C6   A6 67      LDX $67
88C8   86 66      STX $66
88CA   A6 71      LDX $71
88CC   86 67      STX $67
88CE   84 71      STY $71
```

```
88D0   69 08      ADC #$08    Exponentzähler erhöhen
88D2   C9 20      CMP #$20    Schon alle 4 Bytes verschoben ?
88D4   D0 E4      BNE $88BA   Nein, weitertesten

88D6   A9 00      LDA #$00    Alle Bytes der Mantisse = 0
88D8   85 63      STA $63     Also FAC#1 auf 0 setzen
88DA   85 68      STA $68     und Vorzeichen löschen
88DC   60         RTS

88DD   65 58      ADC $58     Vorzeichen auf Exponenten aufaddieren
88DF   85 71      STA $71
88E1   A5 67      LDA $67     Mantisse von FAC#1 und FAC#2 addieren
88E3   65 6E      ADC $6E
88E5   85 67      STA $67
88E7   A5 66      LDA $66
88E9   65 6D      ADC $6D
88EB   85 66      STA $66
88ED   A5 65      LDA $65
88EF   65 6C      ADC $6C
88F1   85 65      STA $65
88F3   A5 64      LDA $64
88F5   65 6B      ADC $6B
88F7   85 64      STA $64
88F9   4C 15 89   JMP $8915   Exponent korrigieren

88FC   69 01      ADC #$01    Exponent erhöhen
88FE   06 71      ASL $71     Bit aus Rundungsbyte ins C-Flag
8900   26 67      ROL $67     Mantissenbytes alle
8902   26 66      ROL $66     um ein Bit verschieben
8904   26 65      ROL $65
8906   26 64      ROL $64
8908   10 F2      BPL $88FC   Weiter bis MSB gesetzt
890A   38         SEC
890B   E5 63      SBC $63     Exponent von FAC#1 abziehen
890D   B0 C7      BCS $88D6   Wenn Ergebnis ohne Übertrag, FAC#1 = 0
890F   49 FF      EOR #$FF    Zweierkomplement bilden
8911   69 01      ADC #$01
8913   85 63      STA $63     und neuen Exponenten speichern

8915   90 0E      BCC $8925   Wenn kein Übertrag, Ende
8917   E6 63      INC $63     Exponent erhöhen
8919   F0 42      BEQ $895D   Wenn zu groß, Fehler
891B   66 64      ROR $64     Mantissenbits alle
891D   66 65      ROR $65     um ein Bit verschieben
891F   66 66      ROR $66
8921   66 67      ROR $67
8923   66 71      ROR $71     Rundungsbyte korrigieren
```

```
8925   60         RTS

****************************** Mantisse von FAC#1 invertieren

8926   A5 68      LDA $68
8928   49 FF      EOR #$FF
892A   85 68      STA $68
892C   A5 64      LDA $64
892E   49 FF      EOR #$FF
8930   85 64      STA $64
8932   A5 65      LDA $65
8934   49 FF      EOR #$FF
8936   85 65      STA $65
8938   A5 66      LDA $66
893A   49 FF      EOR #$FF
893C   85 66      STA $66
893E   A5 67      LDA $67
8940   49 FF      EOR #$FF
8942   85 67      STA $67
8944   A5 71      LDA $71       Rundungsbyte invertieren
8946   49 FF      EOR #$FF
8948   85 71      STA $71       und wieder setzen
894A   E6 71      INC $71       1 zu Mantisse addieren
894C   D0 0E      BNE $895C     zur korrekten Zweierkomplementbildung
894E   E6 67      INC $67       Mantissenbyte erhöhen
8950   D0 0A      BNE $895C     Kein Übertrag, Ende
8952   E6 66      INC $66       Mantissenbyte erhöhen
8954   D0 06      BNE $895C     Kein Übertrag, Ende
8956   E6 65      INC $65       Mantissenbyte erhöhen
8958   D0 02      BNE $895C     Kein Übertrag, Ende
895A   E6 64      INC $64       Mantisssenbyte erhöhen
895C   60         RTS

****************************** 'OVERFLOW'

895D   A2 0F      LDX #$0F      'OVERFLOW'
895F   4C 3C 4D   JMP $4D3C     Fehler ausgeben

****************************** Rechtsverschieben eines Registers

8962   A2 27      LDX #$27      Offset auf Hilfsregister setzen
8964   B4 04      LDY $04,X     Rundungsbyte laden
8966   84 71      STY $71       und setzen
8968   B4 03      LDY $03,X     Mantisse um ein Byte verschieben
896A   94 04      STY $04,X
896C   B4 02      LDY $02,X
896E   94 03      STY $03,X
8970   B4 01      LDY $01,X
```

```
8972   94 02      STY $02,X
8974   AC DF 03   LDY $03DF    Überlaufbyte laden
8977   94 01      STY $01,X    und in Mantisse setzen
8979   69 08      ADC #$08     Exponent korrigieren
897B   30 E7      BMI $8964    Wenn negativ, weiterschieben
897D   F0 E5      BEQ $8964    Wenn Null, weiterschieben
897F   E9 08      SBC #$08     Ergebnis umrechnen
8981   A8         TAY          und als Zähler laden
8982   A5 71      LDA $71      Rundungsbyte laden
8984   B0 14      BCS $899A    Wenn kein Übertrag, Ende
8986   16 01      ASL $01,X    Mantisse korrigieren
8988   90 02      BCC $898C    Kein Übertrag, Skip
898A   F6 01      INC $01,X    Mantissenbyte erhöhen
898C   76 01      ROR $01,X    Alle Mantissenbytes
898E   76 01      ROR $01,X    um ein Bit verschieben
8990   76 02      ROR $02,X
8992   76 03      ROR $03,X
8994   76 04      ROR $04,X
8996   6A         ROR          Rundungsbyte auch anpassen
8997   C8         INY          Zähler schon Null ?
8998   D0 EC      BNE $8986    Nein, weiterschieben
899A   18         CLC
899B   60         RTS

****************************** Konstanten für LOG

899C   81 00 00 00 00          1
89A1   03                      3 = Polynomgrad, 4 Koeffizienten
89A2   7F 5E 56 CB 79          .434255942
89A7   80 13 9B 0B 64          .576584541
89AC   80 76 38 93 16          .961800759
89B1   82 38 AA 3B 20          2.88539007
89B6   80 35 04 F3 34          .707106781 = 1/SQR(2)
89BB   81 35 04 F3 34          1.41421356 = SQR(2)
89C0   80 80 00 00 00          -.5
89C5   80 31 72 17 F8          .693147181 = LOG(2)

****************************** BASIC-Funktion LOG

89CA   20 57 8C   JSR $8C57    Vorzeichen holen
89CD   F0 02      BEQ $89D1    Wenn 0, Ende
89CF   10 03      BPL $89D4    Wenn positiv, Skip
89D1   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'
89D4   A5 63      LDA $63      Exponent
89D6   E9 7F      SBC #$7F     normalisieren
89D8   48         PHA          und retten
89D9   A9 80      LDA #$80     Zahl in Bereich .5 bis 1 bringen
89DB   85 63      STA $63      Exponent setzen
```

```
89DD   A9 B6      LDA #$B6    Zeiger auf 1/SQR(2)
89DF   A0 89      LDY #$89    ab $89B6 setzen
89E1   20 12 8A   JSR $8A12   Konstante zu FAC#1 addieren
89E4   A9 BB      LDA #$BB    Zeiger auf SQR(2)
89E6   A0 89      LDY #$89    ab $89BB setzen
89E8   20 1E 8A   JSR $8A1E   SQR(2) durch FAC#1 dividieren
89EB   A9 9C      LDA #$9C    Zeiger auf 1
89ED   A0 89      LDY #$89    ab $899C setzen
89EF   20 18 8A   JSR $8A18   1 - FAC#1
89F2   A9 A1      LDA #$A1    Zeiger auf Polynomkoeffizienten
89F4   A0 89      LDY #$89    ab $89A1 setzen
89F6   20 86 90   JSR $9086   Polynomberechnung
89F9   A9 C0      LDA #$C0    Zeiger auf -.5
89FB   A0 89      LDY #$89    ab $89C0 setzen
89FD   20 12 8A   JSR $8A12   Zu FAC#1 addieren
8A00   68         PLA         Exponent wieder holen
8A01   20 B0 8D   JSR $8DB0   FAC#1 = 2*FAC#1
8A04   A9 C5      LDA #$C5    Zeiger auf LOG(2)
8A06   A0 89      LDY #$89    ab $89C5 setzen
8A08   20 89 8A   JSR $8A89   FAC#2 mit Konstante laden
8A0B   4C 27 8A   JMP $8A27   FAC#1 = FAC#2 * FAC#1

******************************* FAC#1 = FAC#1 + 0.5

8A0E   A9 76      LDA #$76    Zeiger auf Konstante 0.5
8A10   A0 8F      LDY #$8F    ab $8F76 setzen
8A12   20 89 8A   JSR $8A89   Konstante (A)/(Y) in FAC#2
8A15   4C 48 88   JMP $8848   FAC#1 = FAC#2 + FAC#1

******************************* FAC#1 = Konstante (A)/(Y) - FAC#1

8A18   20 89 8A   JSR $8A89   Konstante (A)/(Y) in FAC#2
8A1B   4C 31 88   JMP $8831   FAC#1 = FAC#2 - FAC#1

******************************* FAC#1 = Konstante (A)/(Y) / FAC#1

8A1E   20 89 8A   JSR $8A89   Konstante (A)/(Y) in FAC#2
8A21   4C 4C 8B   JMP $8B4C   FAC#1 = FAC#2 / FAC#1

******************************* FAC#1 = Konstante (A)/(Y) * FAC#1

8A24   20 B4 8A   JSR $8AB4   FAC#2 = Konstante (A)/(Y) Bank 1

******************************* FAC#1 = FAC#2 * FAC#1

8A27   D0 03      BNE $8A2C   Wenn <> 0, Skip
8A29   4C 88 8A   JMP $8A88   RTS
8A2C   20 EC 8A   JSR $8AEC   Exponent berechnen
```

```
8A2F   A9 00      LDA #$00     Mantisse des
8A31   85 28      STA $28      Ergebnisses löschen
8A33   85 29      STA $29
8A35   85 2A      STA $2A
8A37   85 2B      STA $2B
8A39   A5 71      LDA $71      Rundungsbyte laden
8A3B   20 55 8A   JSR $8A55    BIT-Multiplikation
8A3E   A5 67      LDA $67      Mantissenbyte laden
8A40   20 55 8A   JSR $8A55    BIT-Multiplikation
8A43   A5 66      LDA $66      Mantissenbyte laden
8A45   20 55 8A   JSR $8A55    BIT-Multiplikation
8A48   A5 65      LDA $65      Mantissenbyte laden
8A4A   20 55 8A   JSR $8A55    BIT-Multiplikation
8A4D   A5 64      LDA $64      Mantissenbyte laden
8A4F   20 5B 8A   JSR $8A5B    BIT-Multiplikation
8A52   4C C1 8B   JMP $8BC1    Ergebnis in FAC#1

****************************** BIT-Multiplikation

8A55   D0 04      BNE $8A5B    Wenn Byte <> 0, Skip
8A57   38         SEC          Addition mit Übertrag
8A58   4C 62 89   JMP $8962    Ergebnis rechtsverschieben
8A5B   4A         LSR          Ein Bit in Carry
8A5C   09 80      ORA #$80     und MSB als Endeflag setzen
8A5E   A8         TAY          Byte retten
8A5F   90 19      BCC $8A7A    Wenn Bit nicht gesetzt, Skip
8A61   18         CLC          Mantisse von FAC#2 auf
8A62   A5 2B      LDA $2B      Ergebnisspeicher aufaddieren
8A64   65 6E      ADC $6E
8A66   85 2B      STA $2B
8A68   A5 2A      LDA $2A
8A6A   65 6D      ADC $6D
8A6C   85 2A      STA $2A
8A6E   A5 29      LDA $29
8A70   65 6C      ADC $6C
8A72   85 29      STA $29
8A74   A5 28      LDA $28
8A76   65 6B      ADC $6B
8A78   85 28      STA $28
8A7A   66 28      ROR $28      Mantisse des Ergebnisses
8A7C   66 29      ROR $29      um ein Bit verschieben
8A7E   66 2A      ROR $2A
8A80   66 2B      ROR $2B
8A82   66 71      ROR $71      Rundungsbyte anpassen
8A84   98         TYA          Multiplikationsbyte wieder holen
8A85   4A         LSR          Nächstes Bit in C-Flag
8A86   D0 D6      BNE $8A5E    Wenn Byte noch nicht Null, nochmal
```

```
8A88   60          RTS

******************************* FAC#2 = Konstante (A)/(Y)

8A89   85 24       STA $24        Adresse Low-Byte setzen
8A8B   84 25       STY $25        Adresse High-Byte setzen
8A8D   A0 04       LDY #$04       Ein Real-Wert hat 5 Bytes
8A8F   B1 24       LDA ($24),Y    Mantissenbyte laden
8A91   85 6E       STA $6E        und setzen
8A93   88          DEY            Offset erniedrigen
8A94   B1 24       LDA ($24),Y    Mantissenbyte laden
8A96   85 6D       STA $6D        und setzen
8A98   88          DEY            Offset erniedrigen
8A99   B1 24       LDA ($24),Y    Mantissenbyte laden
8A9B   85 6C       STA $6C        und setzen
8A9D   88          DEY            Offset erniedrigen
8A9E   B1 24       LDA ($24),Y    Vorzeichenbyte laden
8AA0   85 6F       STA $6F        und setzen
8AA2   45 68       EOR $68        Vorzeichenvergleich durchführen
8AA4   85 70       STA $70        Vorzeichenvergleich setzen
8AA6   A5 6F       LDA $6F        Vorzeichenbyte laden
8AA8   09 80       ORA #$80       Bit 7 auf 1 setzen
8AAA   85 6B       STA $6B        und Mantissenbyte setzen
8AAC   88          DEY            Offset erniedrigen
8AAD   B1 24       LDA ($24),Y    Exponent laden
8AAF   85 6A       STA $6A        und setzen
8AB1   A5 63       LDA $63        Exponent von FAC#1 laden
8AB3   60          RTS

******************************* FAC#2 = Konstante (A)/(Y) Bank 1

8AB4   85 24       STA $24        Adresse Low-Byte setzen
8AB6   84 25       STY $25        Adresse High-Byte setzen
8AB8   AD 00 FF    LDA $FF00      Bankvorwahl laden
8ABB   48          PHA            und retten
8ABC   A0 04       LDY #$04       Ein Real-Wert hat 5 Bytes
8ABE   20 B7 03    JSR $03B7      Mantissenbyte laden
8AC1   85 6E       STA $6E        und setzen
8AC3   88          DEY            Offset erniedrigen
8AC4   20 B7 03    JSR $03B7      Mantissenbyte laden
8AC7   85 6D       STA $6D        und setzen
8AC9   88          DEY            Offset erniedrigen
8ACA   20 B7 03    JSR $03B7      Mantissenbyte laden
8ACD   85 6C       STA $6C        und setzen
8ACF   88          DEY            Offset erniedrigen
8AD0   20 B7 03    JSR $03B7      Vorzeichenbyte laden
8AD3   85 6F       STA $6F        und setzen
8AD5   45 68       EOR $68        Vorzeichenvergleich durchführen
```

```
8AD7   85 70      STA $70      Vorzeichenvergleich setzen
8AD9   A5 6F      LDA $6F      Vorzeichen laden
8ADB   09 80      ORA #$80     Bit 7 setzen
8ADD   85 6B      STA $6B      und mantissenbyte setzen
8ADF   88         DEY          Offset erniedrigen
8AE0   20 B7 03   JSR $03B7    Exponent laden
8AE3   85 6A      STA $6A      und setzen
8AE5   68         PLA          Bankvorgabe holen
8AE6   8D 00 FF   STA $FF00    Bank wieder setzen
8AE9   A5 63      LDA $63      Exponent von FAC#1 laden
8AEB   60         RTS

****************************** Exponenten berechnen

8AEC   A5 6A      LDA $6A      Exponent FAC#2 laden
8AEE   F0 1F      BEQ $8B0F    Wenn FAC#2 = 0, Ende
8AF0   18         CLC          Exponent von FAC#2
8AF1   65 63      ADC $63      zu Exponent von FAC#1 addieren
8AF3   90 04      BCC $8AF9    Kein Überlauf, Skip
8AF5   30 1D      BMI $8B14    Wenn Überlauf, Fehler
8AF7   18         CLC
8AF8   2C         .BYTE $2C    Nächsten Befehl überlesen
8AF9   10 14      BPL $8B0F    Wenn Ergebnis < $80, Rücksprung
8AFB   69 80      ADC #$80     Neuen Exponenten anpassen
8AFD   85 63      STA $63      und Exponent FAC#1 setzen
8AFF   D0 03      BNE $8B04    Wenn Wert <> 0, Skip
8B01   4C DA 88   JMP $88DA    FAC#1 = 0 (Exponent auf 0)
8B04   A5 70      LDA $70      Vorzeichenvergleich laden
8B06   85 68      STA $68      und Vorzeichen FAC#1 setzen
8B08   60         RTS

****************************** Test auf Überlauf

8B09   A5 68      LDA $68      Vorzeichenbyte FAC#1 laden
8B0B   49 FF      EOR #$FF     Wert vertretbar ?
8B0D   30 05      BMI $8B14    Nein, Fehler
8B0F   68         PLA          Rücksprungadresse vom
8B10   68         PLA          Stapel löschen
8B11   4C D6 88   JMP $88D6    FAC#1 = 0
8B14   4C 5D 89   JMP $895D    'OVERFLOW'

****************************** FAC#1 = FAC#1 * 10

8B17   20 38 8C   JSR $8C38    FAC#1 runden und in FAC#2
8B1A   AA         TAX          FAC#1 = 0 ?
8B1B   F0 10      BEQ $8B2D    Ja, Ende
8B1D   18         CLC          Exponent + 2 (= Wert * 4)
8B1E   69 02      ADC #$02
```

```
8B20   B0 F2      BCS $8B14    Wenn zu groß, Fehler
8B22   A2 00      LDX #$00     Flag für Vorzeichenvergleich
8B24   86 70      STX $70      löschen
8B26   20 55 88   JSR $8855    FAC#1 = FAC#2 + FAC#1 (= Wert * 5)
8B29   E6 63      INC $63      Exponent + 1 (= Wert * 10)
8B2B   F0 E7      BEQ $8B14    Wenn zu groß, Fehler
8B2D   60         RTS

****************************** Konstante 10

8B2E   84 20 00 00 00          10

****************************** 'DIVISION BY ZERO'

8B33   A2 14      LDX #$14     'DIVISION BY ZERO'
8B35   4C 3C 4D   JMP $4D3C    Fehler ausgeben

****************************** FAC#1 = FAC#1 / 10

8B38   20 38 8C   JSR $8C38    FAC#1 runden und nach FAC#2
8B3B   A9 2E      LDA #$2E     Zeiger auf Konstante 10
8B3D   A0 8B      LDY #$8B     ab $8B2E setzen
8B3F   A2 00      LDX #$00     Vorzeichenvergleich löschen
8B41   86 70      STX $70
8B43   20 D4 8B   JSR $8BD4    Konstante in FAC#1
8B46   4C 4C 8B   JMP $8B4C    FAC#1 = FAC#2 / FAC#1

****************************** FAC#1 = Konstante (A)/(Y) / FAC#1

8B49   20 B4 8A   JSR $8AB4    Konstante (A)/(Y) in FAC#2

****************************** FAC#1 = FAC#2 / FAC#1

8B4C   F0 E5      BEQ $8B33    Wenn FAC#1 = 0, Fehler
8B4E   20 47 8C   JSR $8C47    FAC#1 runden
8B51   A9 00      LDA #$00
8B53   38         SEC
8B54   E5 63      SBC $63      Exponent FAC#1 negieren
8B56   85 63      STA $63      und setzen
8B58   20 EC 8A   JSR $8AEC    Exponent berechnen
8B5B   E6 63      INC $63      Exponentenüberlauf ?
8B5D   F0 B5      BEQ $8B14    Ja, Fehler
8B5F   A2 FC      LDX #$FC     Bytezähler setzen
8B61   A9 01      LDA #$01     Bitzähler setzen
8B63   A4 6B      LDY $6B      FAC#2 mit FAC#1 byteweise vergleichen
8B65   C4 64      CPY $64
8B67   D0 10      BNE $8B79    Wenn ungleich, Skip
8B69   A4 6C      LDY $6C
```

```
8B6B   C4 65      CPY $65
8B6D   D0 0A      BNE $8B79   Wenn ungleich, Skip
8B6F   A4 6D      LDY $6D
8B71   C4 66      CPY $66
8B73   D0 04      BNE $8B79   Wenn ungleich, Skip
8B75   A4 6E      LDY $6E
8B77   C4 67      CPY $67
8B79   08         PHP         Status retten
8B7A   2A         ROL         Bitzähler verschieben
8B7B   90 09      BCC $8B86   Wenn Bit nicht gesetzt, Skip
8B7D   E8         INX         Letztes Byte verarbeitet ?
8B7E   95 2B      STA $2B,X   Ergebnisbyte setzen
8B80   F0 32      BEQ $8BB4   Bei letztem Byte, Skip
8B82   10 34      BPL $8BB8   Wenn Ende, Skip
8B84   A9 01      LDA #$01    Bitzähler neu setzen
8B86   28         PLP         FAC#2 >= FAC#1 ?
8B87   B0 0E      BCS $8B97   Ja, Skip
8B89   06 6E      ASL $6E     Mantisse FAC#2 verschieben
8B8B   26 6D      ROL $6D
8B8D   26 6C      ROL $6C
8B8F   26 6B      ROL $6B
8B91   B0 E6      BCS $8B79   Wenn MSB gesetzt war, weitertesten
8B93   30 CE      BMI $8B63   Wenn MSB nun gesetzt, neu vergleichen
8B95   10 E2      BPL $8B79   Unbedingter Sprung

8B97   A8         TAY         Bitzähler retten
8B98   A5 6E      LDA $6E     Mantissen von FAC#1 von FAC#2
8B9A   E5 67      SBC $67     subtrahieren
8B9C   85 6E      STA $6E
8B9E   A5 6D      LDA $6D
8BA0   E5 66      SBC $66
8BA2   85 6D      STA $6D
8BA4   A5 6C      LDA $6C
8BA6   E5 65      SBC $65
8BA8   85 6C      STA $6C
8BAA   A5 6B      LDA $6B
8BAC   E5 64      SBC $64
8BAE   85 6B      STA $6B
8BB0   98         TYA         Bitzähler wieder laden
8BB1   4C 89 8B   JMP $8B89   Mantisse FAC#2 verschieben

8BB4   A9 40      LDA #$40    Bitzähler passend für letztes Byte setzen
8BB6   D0 CE      BNE $8B86   Unbedingter Sprung

8BB8   0A         ASL         (A) = (A) * 64
8BB9   0A         ASL
8BBA   0A         ASL
8BBB   0A         ASL
```

```
8BBC   0A          ASL
8BBD   0A          ASL
8BBE   85 71       STA $71         Rundungsbyte setzen
8BC0   28          PLP             Status wieder holen

****************************** Hilfsregister in FAC#1

8BC1   A5 28       LDA $28         Mantissenbytes aus Hilfsregister
8BC3   85 64       STA $64         in FAC#1 kopieren
8BC5   A5 29       LDA $29
8BC7   85 65       STA $65
8BC9   A5 2A       LDA $2A
8BCB   85 66       STA $66
8BCD   A5 2B       LDA $2B
8BCF   85 67       STA $67
8BD1   4C B6 88    JMP $88B6       FAC#1 linksbündig machen

****************************** Konstante (A)/(Y) in FAC#1

8BD4   85 24       STA $24         Adresse Low-Byte setzen
8BD6   84 25       STY $25         Adresse High-Byte setzen
8BD8   A0 04       LDY #$04        Ein Real-Wert hat 5 Bytes
8BDA   B1 24       LDA ($24),Y     Mantissenbyte laden
8BDC   85 67       STA $67         und setzen
8BDE   88          DEY             Offset erniedrigen
8BDF   B1 24       LDA ($24),Y     Mantissenbyte laden
8BE1   85 66       STA $66         und setzen
8BE3   88          DEY             Offset erniedrigen
8BE4   B1 24       LDA ($24),Y     Mantissenbyte laden
8BE6   85 65       STA $65         und setzen
8BE8   88          DEY             Offset erniedrigen
8BE9   B1 24       LDA ($24),Y     Vorzeichenbyte laden
8BEB   85 68       STA $68         und setzen
8BED   09 80       ORA #$80        Bit 7 setzen
8BEF   85 64       STA $64         und Mantissenbyte setzen
8BF1   88          DEY             Offset erniedrigen
8BF2   B1 24       LDA ($24),Y     Exponent laden
8BF4   85 63       STA $63         und setzen
8BF6   84 71       STY $71         Rundungsbyte löschen ((Y)=0)
8BF8   60          RTS

****************************** FAC#1 in Akku 4

8BF9   A2 5E       LDX #$5E        Offset auf Akku 4
8BFB   2C          .BYTE $2C       Nächsten Befehl überlesen

****************************** FAC#1 in Akku 3
```

```
8BFC   A2 59      LDX #$59     Offset auf Akku 3
8BFE   A0 00      LDY #$00     High-Byte löschen

******************************* FAC#1 nach (X)/(Y) mit Rundung

8C00   20 47 8C   JSR $8C47    FAC#1 runden

******************************* FAC#1 nach (X)/(Y)

8C03   86 24      STX $24      Adresse Low-Byte setzen
8C05   84 25      STY $25      Adresse High-Byte setzen
8C07   A0 04      LDY #$04     Ein Real-Wert hat 5 Bytes
8C09   A5 67      LDA $67      Mantissenbyte laden
8C0B   91 24      STA ($24),Y  und speichern
8C0D   88         DEY          Offset erniedrigen
8C0E   A5 66      LDA $66      Mantissenbyte laden
8C10   91 24      STA ($24),Y  und speichern
8C12   88         DEY          Offset erniedrigen
8C13   A5 65      LDA $65      Mantissenbyte laden
8C15   91 24      STA ($24),Y  und speichern
8C17   88         DEY          Offset erniedrigen
8C18   A5 68      LDA $68      Vorzeichen laden
8C1A   09 7F      ORA #$7F     Aus Vorzeichen das
8C1C   25 64      AND $64      Speicherformat erzeugen
8C1E   91 24      STA ($24),Y  und Vorzeichen speichern
8C20   88         DEY          Offset erniedrigen
8C21   A5 63      LDA $63      Exponent laden
8C23   91 24      STA ($24),Y  und speichern
8C25   84 71      STY $71      Rundungsbyte löschen ((Y)=0)
8C27   60         RTS

******************************* FAC#2 in FAC#1

8C28   A5 6F      LDA $6F      Vorzeichen FAC#2 laden
8C2A   85 68      STA $68      und Vorzeichen FAC#1 setzen
8C2C   A2 05      LDX #$05     Ein Real-Wert hat 5 Bytes
8C2E   B5 69      LDA $69,X    Byte aus FAC#2 laden
8C30   95 62      STA $62,X    und in FAC#1 kopieren
8C32   CA         DEX          Alle Bytes kopiert ?
8C33   D0 F9      BNE $8C2E    Nein, nochmal
8C35   86 71      STX $71      Rundungsbyte löschen ((X)=0)
8C37   60         RTS

******************************* FAC#1 in FAC#2 mit Rundung

8C38   20 47 8C   JSR $8C47    FAC#1 runden

******************************* FAC#1 in FAC#2
```

```
8C3B   A2 06      LDX #$06     Ein Real-Wert hat 5 Bytes + Vorzeichen
8C3D   B5 62      LDA $62,X    Byte aus FAC#1 laden
8C3F   95 69      STA $69,X    und in FAC#2 kopieren
8C41   CA         DEX          Alle Bytes kopiert ?
8C42   D0 F9      BNE $8C3D    Nein, nochmal
8C44   86 71      STX $71      Rundungsbyte löschen
8C46   60         RTS

****************************** FAC#1 runden

8C47   A5 63      LDA $63      FAC#1 = 0 ?
8C49   F0 FB      BEQ $8C46    Ja, Ende
8C4B   06 71      ASL $71      Rundungsstelle > $7F ?
8C4D   90 F7      BCC $8C46    Nein, Ende
8C4F   20 4E 89   JSR $894E    Wert um 1 erhöhen
8C52   D0 F2      BNE $8C46    Wenn nun fertig, Ende
8C54   4C 17 89   JMP $8917    Rechtsverschieben und Exponent erhöhen

****************************** Vorzeichen von FAC#1 holen

8C57   A5 63      LDA $63      FAC#1 = 0 ?
8C59   F0 09      BEQ $8C64    Ja, Ende
8C5B   A5 68      LDA $68      Vorzeichenbyte laden
8C5D   2A         ROL          Vorzeichenbit in C-Flag schieben
8C5E   A9 FF      LDA #$FF     FAC#1 ist negativ
8C60   B0 02      BCS $8C64    Wenn wirklich negativ, Ende
8C62   A9 01      LDA #$01     FAC#1 ist positiv
8C64   60         RTS

****************************** BASIC-Funktion SGN

8C65   20 57 8C   JSR $8C57    Vorzeichen holen
8C68   85 64      STA $64      in Low-Byte
8C6A   A9 00      LDA #$00     High-Byte auf 0
8C6C   85 65      STA $65      setzen
8C6E   A2 88      LDX #$88     Exponent setzen

****************************** Integerwert in FAC#1

8C70   A5 64      LDA $64      Vorzeichenbyte laden
8C72   49 FF      EOR #$FF     und invertieren
8C74   2A         ROL          und Vorzeichenbit in Carry

****************************** Mit SEC,LDX #$90 Adreßwert in FAC#1

8C75   A9 00      LDA #$00
8C77   85 67      STA $67      FAC#1-Stellen löschen
```

```
8C79   85 66      STA $66
8C7B   86 63      STX $63        Exponent setzen
8C7D   85 71      STA $71        Rundungsbyte löschen
8C7F   85 68      STA $68        FAC#1-Stelle löschen
8C81   4C B1 88   JMP $88B1

****************************** BASIC-Funktion ABS

8C84   46 68      LSR $68        Vorzeichen löschen
8C86   60         RTS

****************************** Vergleich Konstante (A)/(Y) mit FAC#1

8C87   85 26      STA $26        Adresse Low-Byte setzen
8C89   84 27      STY $27        Adresse High-Byte setzen
8C8B   A0 00      LDY #$00       Offset auf 0
8C8D   B1 26      LDA ($26),Y    Exponenten der Konstante laden
8C8F   C8         INY            Offset erhöhen
8C90   AA         TAX            Konstante = 0 ?
8C91   F0 C4      BEQ $8C57      Ja, Vorzeichen von FAC#1 ist Ergebnis
8C93   B1 26      LDA ($26),Y    Verschiedene Vorzeichen ?
8C95   45 68      EOR $68
8C97   30 C2      BMI $8C5B      Ja, Vorzeichen verarbeiten
8C99   E4 63      CPX $63        Exponenten gleich ?
8C9B   D0 21      BNE $8CBE      Nein, Skip
8C9D   B1 26      LDA ($26),Y    FAC#1 und Konstante byteweise vergleichen
8C9F   09 80      ORA #$80       FAC-Format erzeugen
8CA1   C5 64      CMP $64        Byte gleich ?
8CA3   D0 19      BNE $8CBE      Nein, Skip
8CA5   C8         INY            Offset erhöhen
8CA6   B1 26      LDA ($26),Y    Byte laden
8CA8   C5 65      CMP $65        Byte gleich ?
8CAA   D0 12      BNE $8CBE      Nein, Skip
8CAC   C8         INY            Offst erhöhen
8CAD   B1 26      LDA ($26),Y    Byte laden
8CAF   C5 66      CMP $66        Byte gleich ?
8CB1   D0 0B      BNE $8CBE      Nein, Skip
8CB3   C8         INY            Offset erhöhen
8CB4   A9 7F      LDA #$7F       Vorgabe für Rundungsbyte laden
8CB6   C5 71      CMP $71        Rundungsbyte gleich Vorgabe ?
8CB8   B1 26      LDA ($26),Y    Unterstes Mantissenbyte laden
8CBA   E5 67      SBC $67        gleich FAC#1 Mantissenbyte ?
8CBC   F0 2A      BEQ $8CE8      Ja, Ende
8CBE   A5 68      LDA $68        Vorzeichen FAC#1 laden
8CC0   90 02      BCC $8CC4      Wenn positiv, Skip
8CC2   49 FF      EOR #$FF       Vorzeichenwert invertieren
8CC4   4C 5D 8C   JMP $8C5D      Ergebnisflags setzen
```

```
****************************** Umwandlung Fließkomma in Integer

8CC7   A5 63      LDA $63      FAC#1 = 0 ?
8CC9   F0 4D      BEQ $8D18    Ja, Ergebnis auf 0
8CCB   38         SEC          Exponent - Grundwert $80 - 32
8CCC   E9 A0      SBC #$A0     ergibt Exponentbasis 0, Zahl durch 65536
8CCE   24 68      BIT $68      Vorzeichen testen
8CD0   10 0A      BPL $8CDC    Wenn positiv, Skip
8CD2   AA         TAX          Neuen Exponenten retten
8CD3   A9 FF      LDA #$FF     Überlaufkennung passend
8CD5   8D DF 03   STA $03DF    setzen
8CD8   20 2C 89   JSR $892C    Mantisse invertieren
8CDB   8A         TXA          Neuen Exponenten wieder laden
8CDC   A2 63      LDX #$63
8CDE   C9 F9      CMP #$F9     Neuer Exponent erfordert Änderung ?
8CE0   10 07      BPL $8CE9    Ja, Skip
8CE2   20 79 89   JSR $8979    FAC#1 rechtsverschieben
8CE5   8C DF 03   STY $03DF    Überlaufkennung passend setzen
8CE8   60         RTS

8CE9   A8         TAY          Neuen Exponenten retten
8CEA   A5 68      LDA $68      Vorzeichenbyte laden
8CEC   29 80      AND #$80     Voryeichen isolieren
8CEE   46 64      LSR $64      Mantissenbyte rechtsverschieben
8CF0   05 64      ORA $64      Vorzeichen in Mantisse setzen
8CF2   85 64      STA $64      und mantissenbyte wieder setzen
8CF4   20 90 89   JSR $8990    FAC#1 bitweise rechtsverschieben
8CF7   8C DF 03   STY $03DF    Überlaufkennung setzen
8CFA   60         RTS

****************************** BASIC-Funktion INT

8CFB   A5 63      LDA $63      Exponent FAC#1 laden
8CFD   C9 A0      CMP #$A0     FAC#1 ist ganze Zahl ?
8CFF   B0 20      BCS $8D21    Ja, Ende
8D01   20 68 AA   JSR $AA68    FAC#1 in Integer
8D04   84 71      STY $71      Rundungsbyte setzen
8D06   A5 68      LDA $68      Vorzeichenbyte laden
8D08   84 68      STY $68      Byte anders setzen
8D0A   49 80      EOR #$80     Vorzeichen invertieren
8D0C   2A         ROL          und in C-Flag schieben
8D0D   A9 A0      LDA #$A0     Exponent für ganze Zahl
8D0F   85 63      STA $63      setzen
8D11   A5 67      LDA $67      Mantissenbyte laden
8D13   85 09      STA $09      und umkopieren
8D15   4C B1 88   JMP $88B1    FAC#1 linksbündig machen

8D18   85 64      STA $64      Mantisse mit 0 füllen
```

```
8D1A    85 65       STA $65
8D1C    85 66       STA $66
8D1E    85 67       STA $67
8D20    A8          TAY              (Y)=0
8D21    60          RTS

****************************** Umwandlung ASCII in Fließkommaformat

8D22    8E DA 03    STX $03DA        Bank-Flag setzen
8D25    A0 00       LDY #$00
8D27    A2 0A       LDX #$0A         FAC#1 und Stellenzähler löschen
8D29    94 5F       STY $5F,X
8D2B    CA          DEX              Alle Bytes gelöscht ?
8D2C    10 FB       BPL $8D29        Nein, nochmal
8D2E    90 0F       BCC $8D3F        Wenn letztes Zeichen Zahl, Skip
8D30    C9 2D       CMP #$2D         '-' ?
8D32    D0 04       BNE $8D38        Nein, Skip
8D34    86 69       STX $69          Negativflag setzen
8D36    F0 04       BEQ $8D3C        Unbedingter Sprung
8D38    C9 2B       CMP #$2B         '+' ?
8D3A    D0 05       BNE $8D41        Nein, Skip
8D3C    20 F5 8D    JSR $8DF5        Spezial CHRGET
8D3F    90 5B       BCC $8D9C        Wenn Zeichen Zahl, Skip
8D41    C9 2E       CMP #$2E         '.' ?
8D43    F0 2E       BEQ $8D73        Nein, Skip
8D45    C9 45       CMP #$45         'E' ?
8D47    D0 30       BNE $8D79        Nein, Skip
8D49    20 F5 8D    JSR $8DF5        Spezial CHRGET
8D4C    90 17       BCC $8D65        Wenn Zahl, Skip
8D4E    C9 AB       CMP #$AB         Token für '-' ?
8D50    F0 0E       BEQ $8D60        Ja, Flag setzen
8D52    C9 2D       CMP #$2D         '-' ?
8D54    F0 0A       BEQ $8D60        Ja, Flag setzen
8D56    C9 AA       CMP #$AA         Token für '+'
8D58    F0 08       BEQ $8D62        Ja, Skip
8D5A    C9 2B       CMP #$2B         '+' ?
8D5C    F0 04       BEQ $8D62        Ja, Skip
8D5E    D0 07       BNE $8D67        Nein, Skip
8D60    66 62       ROR $62          E-Negativflag setzen
8D62    20 F5 8D    JSR $8DF5        Spezial CHRGET für E-Zahl
8D65    90 5C       BCC $8DC3        Wenn Zahl, Skip
8D67    24 62       BIT $62          E-Negativflag gesetzt ?
8D69    10 0E       BPL $8D79        Nein, Skip
8D6B    A9 00       LDA #$00         E-Zahl negieren
8D6D    38          SEC
8D6E    E5 60       SBC $60
8D70    4C 7B 8D    JMP $8D7B
8D73    66 61       ROR $61          Aufruf durch '.' BIT 7 setzen
```

```
8D75   24 61      BIT $61      BIT 6 gesetzt ? (2. Punkt gelesen)
8D77   50 C3      BVC $8D3C    Nein, weiterlesen

8D79   A5 60      LDA $60      E-Zahl
8D7B   38         SEC
8D7C   E5 5F      SBC $5F      - Nachkommastellen
8D7E   85 60      STA $60      = nötiger Verschub
8D80   F0 12      BEQ $8D94    Wenn 0, Ende
8D82   10 09      BPL $8D8D    Wenn positiv, Skip
8D84   20 38 8B   JSR $8B38    FAC#1 = FAC#1 / 10
8D87   E6 60      INC $60      Fertig ?
8D89   D0 F9      BNE $8D84    Nein, weiter
8D8B   F0 07      BEQ $8D94    Unbedingter Sprung

8D8D   20 17 8B   JSR $8B17    FAC#1 = FAC#1 * 10
8D90   C6 60      DEC $60      Fertig ?
8D92   D0 F9      BNE $8D8D    Nein, weiter
8D94   A5 69      LDA $69      Negativflag gesetzt ?
8D96   30 01      BMI $8D99    Ja, Skip
8D98   60         RTS
8D99   4C FA 8F   JMP $8FFA    FAC#1 = -FAC#1

8D9C   48         PHA          Aufruf durch Ziffer
8D9D   24 61      BIT $61      '.' gelesen ?
8D9F   10 02      BPL $8DA3    Nein, Skip
8DA1   E6 5F      INC $5F      Eine Nachkommastelle mehr
8DA3   20 17 8B   JSR $8B17    FAC#1 = FAC#1 * 10
8DA6   68         PLA          Zeichen holen
8DA7   38         SEC
8DA8   E9 30      SBC #$30     Umwandlung von ASCII in Zahl
8DAA   20 B0 8D   JSR $8DB0    Diese Stelle zu FAC#1 addieren
8DAD   4C 3C 8D   JMP $8D3C    Weitermachen

8DB0   48         PHA          Stellenwert retten
8DB1   20 38 8C   JSR $8C38    FAC#1 in FAC#2 mit Rundung
8DB4   68         PLA          Stellenwert holen
8DB5   20 68 8C   JSR $8C68    Stellenwert in FAC#1
8DB8   A5 6F      LDA $6F      Vorzeichen vergleichen
8DBA   45 68      EOR $68
8DBC   85 70      STA $70      Vorzeichenvergleich setzen
8DBE   A6 63      LDX $63      Exponent FAC#1
8DC0   4C 48 88   JMP $8848    FAC#1 = FAC#2 + FAC#1

8DC3   A5 60      LDA $60      Aufruf durch E-Zahl
8DC5   C9 0A      CMP #$0A     E-Zahl < 10 ?
8DC7   90 09      BCC $8DD2    Ja, Skip
8DC9   A9 64      LDA #$64     Dezimal 100
8DCB   24 62      BIT $62      Vorzeichen gesetzt ?
```

```
8DCD   30 21      BMI $8DF0   Ja, 100 als E-Zahl
8DCF   4C 5D 89   JMP $895D   'OVERFLOW ERROR'
8DD2   0A         ASL         = Wert * 4
8DD3   0A         ASL
8DD4   18         CLC
8DD5   65 60      ADC $60     = Wert * 5
8DD7   0A         ASL         = Wert * 10
8DD8   18         CLC
8DD9   A0 00      LDY #$00    Offset auf 0
8DDB   85 79      STA $79     E-Zahl speichern
8DDD   AD DA 03   LDA $03DA   Zeichen aus Bank 1 ?
8DE0   D0 06      BNE $8DE8   Ja, Skip
8DE2   20 C9 03   JSR $03C9   Aktuelles Zeichen aus Bank 0
8DE5   4C EB 8D   JMP $8DEB
8DE8   20 B7 03   JSR $03B7   Aktuelles Zeichen aus Bank 1
8DEB   65 79      ADC $79     + E-Zahl
8DED   38         SEC
8DEE   E9 30      SBC #$30    Von ASCII in Zahl wandeln
8DF0   85 60      STA $60     und als neue E-Zahl speichern
8DF2   4C 62 8D   JMP $8D62   E-Zahl weitereinlesen

****************************** Spezial CHRGET

8DF5   AD DA 03   LDA $03DA   Zeichen aus Bank 0 ?
8DF8   D0 03      BNE $8DFD   Nein, Skip
8DFA   4C 80 03   JMP $0380   CHRGET
8DFD   E6 24      INC $24     ($24)CHRGET
8DFF   D0 02      BNE $8E03   Kein Übertrag, Skip
8E01   E6 25      INC $25     Übertrag berücksichtigen
8E03   A0 00      LDY #$00    Offset auf 0 setzen
8E05   20 B7 03   JSR $03B7   LDA ($24),Y Bank 1
8E08   C9 3A      CMP #$3A    Trennzeichen ':' ?
8E0A   B0 0A      BCS $8E16   Ja, Ende
8E0C   C9 20      CMP #$20    Space ?
8E0E   F0 ED      BEQ $8DFD   Ja, überlesen
8E10   38         SEC         Zahlentest
8E11   E9 30      SBC #$30
8E13   38         SEC
8E14   E9 D0      SBC #$D0
8E16   60         RTS

****************************** Konstanten für Fließkomma nach ASCII

8E17   9B 3E BC 1F FD         99999999.9
8E1C   9E 6E 6B 27 FD         999999999
8E21   9E 6E 6B 28 00         1E9
```

```
****************************** Zeilennummer bei Fehler ausgeben

8E26   20 81 92   JSR $9281    PRIMM Stringausgabe

****************************** Konstante ' IN '

8E29   20 49 4E 20 00          ' IN '

****************************** Aktuelle Zeilennummer ausgeben

8E2E   A5 3C      LDA $3C      Zeilennummer High laden
8E30   A6 3B      LDX $3B      Zeilennummer Low laden

****************************** Adresse in (X)/(A) ausgeben

8E32   85 64      STA $64      Adresse im High/Low-Format in FAC#1
8E34   86 65      STX $65      setzen
8E36   A2 90      LDX #$90     Exponent setzen
8E38   38         SEC          Vorzeichenwert setzen
8E39   20 75 8C   JSR $8C75    Adreßwert in Fließkommaformat
8E3C   20 44 8E   JSR $8E44    FAC#1 in ASCII wandeln
8E3F   4C E2 55   JMP $55E2    String ausgeben bis 0-Byte

****************************** FAC#1 in ASCII ab $0100

8E42   A0 01      LDY #$01     Offset vorbesetzen
8E44   A9 20      LDA #$20     Space als erstes Zeichen
8E46   24 68      BIT $68      FAC#1 negativ ?
8E48   10 02      BPL $8E4C    Nein, Skip
8E4A   A9 2D      LDA #$2D     '-' als erstes Zeichen
8E4C   99 FF 00   STA $00FF,Y  in Puffer speichern
8E4F   85 68      STA $68      Vorzeichen positiv machen
8E51   84 72      STY $72      Offset retten
8E53   C8         INY          Offset erhöhen
8E54   A9 30      LDA #$30     '0'
8E56   A6 63      LDX $63      FAC#1 = 0 ?
8E58   D0 03      BNE $8E5D    nein, Skip
8E5A   4C 69 8F   JMP $8F69    Puffer mit '0' besetzen und Ende
8E5D   A9 00      LDA #$00     Flag setzen
8E5F   E0 80      CPX #$80     FAC#1 = zwischen .5 und 1 ?
8E61   F0 02      BEQ $8E65    Ja, Skip
8E63   B0 09      BCS $8E6E    Nein, Skip
8E65   A9 21      LDA #$21     Zeiger auf Konstante 1E9
8E67   A0 8E      LDY #$8E     ab $8E21 setzen
8E69   20 08 8A   JSR $8A08    FAC#1 = Konstante * FAC#1
8E6C   A9 F7      LDA #$F7     Stellenverschub auf -9 setzen ($F7 = -9)
8E6E   85 5F      STA $5F
8E70   A9 1C      LDA #$1C     Zeiger auf Konstante 999999999
```

```
8E72   A0 8E      LDY #$8E      ab $8E1C setzen
8E74   20 87 8C   JSR $8C87     Vergleich Konstante mit FAC#1
8E77   F0 1E      BEQ $8E97     Wenn gleich, Skip
8E79   10 12      BPL $8E8D     Wenn größer, Teilung durch 10
8E7B   A9 17      LDA #$17      Zeiger auf Konstante 99999999.9
8E7D   A0 8E      LDY #$8E      ab $8E17 setzen
8E7F   20 87 8C   JSR $8C87     Vergleich Konstante mit FAC#1
8E82   F0 02      BEQ $8E86     Gleich, Skip
8E84   10 0E      BPL $8E94     Wenn größer, Skip zur Rundung
8E86   20 17 8B   JSR $8B17     FAC#1 = FAC#1 * 10
8E89   C6 5F      DEC $5F       Stellenverschub korrigieren
8E8B   D0 EE      BNE $8E7B     Unbedingter Sprung

8E8D   20 38 8B   JSR $8B38     FAC#1 = FAC#1 / 10
8E90   E6 5F      INC $5F       Stellenverschub korrigieren
8E92   D0 DC      BNE $8E70     Unbedingter Sprung

8E94   20 0E 8A   JSR $8A0E     FAC#1 = INT (FAC#1 + .5)
8E97   20 C7 8C   JSR $8CC7     FAC#1 in Integerformat
8E9A   A2 01      LDX #$01
8E9C   A5 5F      LDA $5F
8E9E   18         CLC
8E9F   69 0A      ADC #$0A      FAC#1 < 0.1 ?
8EA1   30 09      BMI $8EAC     Ja, Skip
8EA3   C9 0B      CMP #$0B      FAC#1 > 1E9 ?
8EA5   B0 06      BCS $8EAD     Ja, Skip
8EA7   69 FF      ADC #$FF
8EA9   AA         TAX
8EAA   A9 02      LDA #$02
8EAC   38         SEC
8EAD   E9 02      SBC #$02
8EAF   85 60      STA $60
8EB1   86 5F      STX $5F
8EB3   8A         TXA
8EB4   F0 02      BEQ $8EB8
8EB6   10 13      BPL $8ECB
8EB8   A4 72      LDY $72       Offset holen
8EBA   A9 2E      LDA #$2E      '.' in Puffer
8EBC   C8         INY           Offset erhöhen
8EBD   99 FF 00   STA $00FF,Y   Zeichen in Puffer schreiben
8EC0   8A         TXA           Stelle nach Komma gleich 0 ?
8EC1   F0 06      BEQ $8EC9     Nein, Skip
8EC3   A9 30      LDA #$30      '0' in Puffer
8EC5   C8         INY           Offset erhöhen
8EC6   99 FF 00   STA $00FF,Y   Zeichen in Puffer schreiben
8EC9   84 72      STY $72       Offset retten
8ECB   A0 00      LDY #$00      Offset auf 0 setzen
8ECD   A2 80      LDX #$80      Ziffernzähler setzen
```

```
8ECF   A5 67      LDA $67       Durch Aufaddieren Berechnung der Ziffern
8ED1   18         CLC
8ED2   79 7E 8F   ADC $8F7E,Y
8ED5   85 67      STA $67
8ED7   A5 66      LDA $66
8ED9   79 7D 8F   ADC $8F7D,Y
8EDC   85 66      STA $66
8EDE   A5 65      LDA $65
8EE0   79 7C 8F   ADC $8F7C,Y
8EE3   85 65      STA $65
8EE5   A5 64      LDA $64
8EE7   79 7B 8F   ADC $8F7B,Y
8EEA   85 64      STA $64
8EEC   E8         INX           Ziffernzähler erhöhen
8EED   B0 04      BCS $8EF3     Additionsübertrag, Skip
8EEF   10 DE      BPL $8ECF     Zähler positiv, nochmal
8EF1   30 02      BMI $8EF5     Unbedingter Sprung
8EF3   30 DA      BMI $8ECF     Zähler negativ, nochmal
8EF5   8A         TXA           Zähler als Ziffer laden
8EF6   90 04      BCC $8EFC     Kein Additionsübertrag, Skip
8EF8   49 FF      EOR #$FF      Ziffer positiv machen
8EFA   69 0A      ADC #$0A      + 10
8EFC   69 2F      ADC #$2F      Code für '0' - 1
8EFE   C8         INY           Offset um 4 erhöhen
8EFF   C8         INY
8F00   C8         INY
8F01   C8         INY
8F02   84 49      STY $49       Offset retten
8F04   A4 72      LDY $72       Offset holen
8F06   C8         INY           Offset erhöhen
8F07   AA         TAX           Ziffer in (X) retten
8F08   29 7F      AND #$7F      in ASCII setzen
8F0A   99 FF 00   STA $00FF,Y   Stelle in Puffer
8F0D   C6 5F      DEC $5F       Dezimalpunkt nötig ?
8F0F   D0 06      BNE $8F17     Nein, Skip
8F11   A9 2E      LDA #$2E      '.' in Puffer
8F13   C8         INY           Offset erhöhen
8F14   99 FF 00   STA $00FF,Y   Punkt in Puffer schreiben
8F17   84 72      STY $72       Offset wieder retten
8F19   A4 49      LDY $49       Offset laden
8F1B   8A         TXA           Ziffer in (A)
8F1C   49 FF      EOR #$FF      Wert invertieren
8F1E   29 80      AND #$80      Bit 7 isolieren
8F20   AA         TAX           Bit 7 in (X) retten
8F21   C0 24      CPY #$24      Tabellenende bei FAC#1-Wandlung erreicht
?
8F23   F0 04      BEQ $8F29     Ja, Skip
8F25   C0 3C      CPY #$3C      Tabellenende bei TI$-Wandlung erreicht ?
```

```
8F27   D0 A6      BNE $8ECF    Nein, weitermachen
8F29   A4 72      LDY $72      Offset holen
8F2B   B9 FF 00   LDA $00FF,Y  zeichen aus Puffer holen
8F2E   88         DEY          Offset erniedrigen
8F2F   C9 30      CMP #$30     Letztes Zeichen eine '0' ?
8F31   F0 F8      BEQ $8F2B    Ja, überlesen
8F33   C9 2E      CMP #$2E     ','
8F35   F0 01      BEQ $8F38    Ja, Skip
8F37   C8         INY          Offset korrigieren
8F38   A9 2B      LDA #$2B     '+'
8F3A   A6 60      LDX $60      Exponent nötig ?
8F3C   F0 2E      BEQ $8F6C    Nein, Skip
8F3E   10 08      BPL $8F48    Ja, Skip
8F40   A9 00      LDA #$00     Exponent negieren
8F42   38         SEC
8F43   E5 60      SBC $60
8F45   AA         TAX
8F46   A9 2D      LDA #$2D     '-'
8F48   99 01 01   STA $0101,Y  in Puffer
8F4B   A9 45      LDA #$45     'E' in Puffer
8F4D   99 00 01   STA $0100,Y  schreiben
8F50   8A         TXA          Expnent in (A)
8F51   A2 2F      LDX #$2F     Code für '0' - 1
8F53   38         SEC
8F54   E8         INX          10er Stelle des Exponenten ausrechnen
8F55   E9 0A      SBC #$0A
8F57   B0 FB      BCS $8F54    Kein Übertrag, nochmal
8F59   69 3A      ADC #$3A     in ASCII umrechnen
8F5B   99 03 01   STA $0103,Y  1er Stelle in Puffer
8F5E   8A         TXA          und
8F5F   99 02 01   STA $0102,Y  10er Stelle in Puffer
8F62   A9 00      LDA #$00     Puffer mit 0 abschließen
8F64   99 04 01   STA $0104,Y
8F67   F0 08      BEQ $8F71    Unbedingter Sprung

8F69   99 FF 00   STA $00FF,Y  Puffer mit eigenem Zeichen abschließen

****************************** Puffer mit 0 abschließen

8F6C   A9 00      LDA #$00
8F6E   99 00 01   STA $0100,Y  Puffer abschließen
8F71   A9 00      LDA #$00     Zeiger auf Puffer
8F73   A0 01      LDY #$01     ab $0100 laden
8F75   60         RTS
```

```
****************************** Konstante 0.5 für SQR

8F76   80 00 00 00 00           .5

****************************** Konstanten für Fließkomma nach ASCII

8F7B   FA 0A 1F 00              -100 000 000
8F7F   00 98 96 80                10 000 000
8F83   FF F0 BD C0                -1 000 000
8F87   00 01 86 A0                   100 000
8F8B   FF FF D8 F0                   -10 000
8F8F   00 00 03 E8                     1 000
8F93   FF FF FF 9C                      -100
8F97   00 00 00 0A                        10
8F9B   FF FF FF FF                        -1

****************************** Konstanten für Umwandlung TI nach TI$

8F9F   FF DF 0A 80              -2 160 000
8FA3   00 03 4B C0                 216 000
8FA7   FF FF 73 60                 -36 000
8FAB   00 00 0E 10                   3 600
8FAF   FF FF FD A8                   - 600
8FB3   00 00 00 3C                      60

****************************** BASIC-Funktion SQR

8FB7   20 38 8C   JSR $8C38     FAC#1 in FAC#2 mit Rundung
8FBA   A9 76      LDA #$76      Zeiger auf Konstante 0.5
8FBC   A0 8F      LDY #$8F      ab $8F76 setzen

****************************** FAC#1 = FAC#2 ^ Konstante (A)/(Y)

8FBE   20 D4 8B   JSR $8BD4     Konstante in FAC#1

****************************** FAC#1 = FAC#2 ^ FAC#1

8FC1   F0 70      BEQ $9033     Konstante = 0, Skip
8FC3   A5 6A      LDA $6A       FAC#2 = 0 ?
8FC5   D0 03      BNE $8FCA     Nein, Skip
8FC7   4C D8 88   JMP $88D8     FAC#1 auf 0
8FCA   A2 50      LDX #$50      Zeiger auf Hilfsakku
8FCC   A0 00      LDY #$00      ab $0050 setzen
8FCE   20 00 8C   JSR $8C00     FAC#1 in Hilfsakku
8FD1   A5 6F      LDA $6F       Potenzexponent < 1 ?
8FD3   10 0F      BPL $8FE4     Ja, Skip
8FD5   20 FB 8C   JSR $8CFB     INT-Funktion
8FD8   A9 50      LDA #$50      Zeiger auf Hilfsakku
```

```
8FDA   A0 00      LDY #$00      ab $0050 setzen
8FDC   20 87 8C   JSR $8C87     Vergleich Konstante mit FAC#1
8FDF   D0 03      BNE $8FE4     Wenn ungleich, Skip
8FE1   98         TYA
8FE2   A4 09      LDY $09
8FE4   20 2A 8C   JSR $8C2A     FAC#2 in FAC#1
8FE7   98         TYA
8FE8   48         PHA
8FE9   20 CA 89   JSR $89CA     LOG-Funktion
8FEC   A9 50      LDA #$50      Zeiger auf Hilfsakku
8FEE   A0 00      LDY #$00
8FF0   20 24 8A   JSR $8A24     FAC#1 = Konstante * FAC#1
8FF3   20 33 90   JSR $9033     EXP-Funktion
8FF6   68         PLA
8FF7   4A         LSR
8FF8   90 0A      BCC $9004

****************************** Vorzeichenwechsel FAC#1

8FFA   A5 63      LDA $63       FAC#1 = 0 ?
8FFC   F0 06      BEQ $9004     Ja, Skip
8FFE   A5 68      LDA $68       Vorzeichen invertieren
9000   49 FF      EOR #$FF
9002   85 68      STA $68
9004   60         RTS

****************************** Konstanten für EXP

9005   81 38 AA 3B 29           1.44269504 = 1/LOG(2)
900A   07                       7 = Polynomgrad ergibt 8 Koeffizienten
900B   71 34 58 3E 56           2.14987637E-5
9010   74 16 7E B3 1B           1.4352314E-4
9015   77 2F EE E3 85           1.34226348E-3
901A   7A 1D 84 1C 2A           9.614011701E-3
901F   7C 63 59 58 0A           .0555051269
9024   7E 75 FD E7 C6           .240226385
9029   80 31 72 18 10           .693147186
902E   81 00 00 00 00           1

****************************** BASIC-Funktion EXP

9033   A9 05      LDA #$05      Zeiger auf Konstante 1/LOG(2)
9035   A0 90      LDY #$90      ab $9005 setzen
9037   20 08 8A   JSR $8A08     FAC#1 = Konstante * FAC#1
903A   A5 71      LDA $71       Rundungsbyte laden
903C   69 50      ADC #$50      Rundung erforderlich ?
903E   90 03      BCC $9043     Nein, Skip
9040   20 4F 8C   JSR $8C4F     Mantisse von FAC#1 um 1 erhöhen
```

```
******************************* Umwandlung Fließkomma in Integer

8CC7   A5 63      LDA $63      FAC#1 = 0 ?
8CC9   F0 4D      BEQ $8D18    Ja, Ergebnis auf 0
8CCB   38         SEC          Exponent - Grundwert $80 - 32
8CCC   E9 A0      SBC #$A0     ergibt Exponentbasis 0, Zahl durch 65536
8CCE   24 68      BIT $68      Vorzeichen testen
8CD0   10 0A      BPL $8CDC    Wenn positiv, Skip
8CD2   AA         TAX          Neuen Exponenten retten
8CD3   A9 FF      LDA #$FF     Überlaufkennung passend
8CD5   8D DF 03   STA $03DF    setzen
8CD8   20 2C 89   JSR $892C    Mantisse invertieren
8CDB   8A         TXA          Neuen Exponenten wieder laden
8CDC   A2 63      LDX #$63
8CDE   C9 F9      CMP #$F9     Neuer Exponent erfordert Änderung ?
8CE0   10 07      BPL $8CE9    Ja, Skip
8CE2   20 79 89   JSR $8979    FAC#1 rechtsverschieben
8CE5   8C DF 03   STY $03DF    Überlaufkennung passend setzen
8CE8   60         RTS

8CE9   A8         TAY          Neuen Exponenten retten
8CEA   A5 68      LDA $68      Vorzeichenbyte laden
8CEC   29 80      AND #$80     Voryeichen isolieren
8CEE   46 64      LSR $64      Mantissenbyte rechtsverschieben
8CF0   05 64      ORA $64      Vorzeichen in Mantisse setzen
8CF2   85 64      STA $64      und mantissenbyte wieder setzen
8CF4   20 90 89   JSR $8990    FAC#1 bitweise rechtsverschieben
8CF7   8C DF 03   STY $03DF    Überlaufkennung setzen
8CFA   60         RTS

******************************* BASIC-Funktion INT

8CFB   A5 63      LDA $63      Exponent FAC#1 laden
8CFD   C9 A0      CMP #$A0     FAC#1 ist ganze Zahl ?
8CFF   B0 20      BCS $8D21    Ja, Ende
8D01   20 68 AA   JSR $AA68    FAC#1 in Integer
8D04   84 71      STY $71      Rundungsbyte setzen
8D06   A5 68      LDA $68      Vorzeichenbyte laden
8D08   84 68      STY $68      Byte anders setzen
8D0A   49 80      EOR #$80     Vorzeichen invertieren
8D0C   2A         ROL          und in C-Flag schieben
8D0D   A9 A0      LDA #$A0     Exponent für ganze Zahl
8D0F   85 63      STA $63      setzen
8D11   A5 67      LDA $67      Mantissenbyte laden
8D13   85 09      STA $09      und umkopieren
8D15   4C B1 88   JMP $88B1    FAC#1 linksbündig machen

8D18   85 64      STA $64      Mantisse mit 0 füllen
```

```
******************************                    Polynomberechnung
y=a0+a1*x+a2*x^2+a3*x^3+...

909C    85 72       STA $72        Zeiger auf Polynomwerte
909E    84 73       STY $73        setzen
90A0    20 F9 8B    JSR $8BF9      FAC#1 in Akku 4
90A3    B1 72       LDA ($72),Y    Polynomgrad
90A5    85 69       STA $69        setzen
90A7    A4 72       LDY $72        Zeiger korrigieren
90A9    C8          INY            Zeiger erhöhen
90AA    98          TYA            Übertrag ?
90AB    D0 02       BNE $90AF      Nein, Skip
90AD    E6 73       INC $73        Übertrag berücksichtigen
90AF    85 72       STA $72        Low-Byte des Zeigers setzen
90B1    A4 73       LDY $73        High-Byte in (Y) laden
90B3    20 08 8A    JSR $8A08      FAC#1 = Konstante * FAC#1
90B6    A5 72       LDA $72        Zeiger auf nächste Zahl setzen
90B8    A4 73       LDY $73
90BA    18          CLC
90BB    69 05       ADC #$05       durch Addition von 5
90BD    90 01       BCC $90C0      Kein Übertrag, Skip
90BF    C8          INY            Übertrag berücksichtigen
90C0    85 72       STA $72        Zeiger Low-Byte neu setzen
90C2    84 73       STY $73        Zeiger High-Byte neu setzen
90C4    20 12 8A    JSR $8A12      FAC#1=Konstante + FAC#1 Zeiger auf Akku4
90C9    A0 00       LDY #$00       ab $005E setzen
90CB    C6 69       DEC $69        Alle Koeffizienten abgearbeitet ?
90CD    D0 E4       BNE $90B3      Nein, weitermachen
90CF    60          RTS

****************************** Fehlermeldung ausgeben

90D0    AA          TAX            Fehlernummer = 0 ?
90D1    D0 02       BNE $90D5      Nein, Skip
90D3    A2 1E       LDX #$1E       'BREAK'
90D5    4C 3C 4D    JMP $4D3C      Fehler ausgeben

****************************** BASIC-OPEN

90D8    20 45 A8    JSR $A845      ROMs einschalten
90DB    20 C0 FF    JSR $FFC0      OPEN
90DE    60          RTS

****************************** BASIC-BSOUT

90DF    20 69 92    JSR $9269      BSOUT
90E2    B0 EC       BCS $90D0      Wenn Fehler, Fehlerroutine
```

```
90E4   60         RTS

****************************** BASIC-BASIN

90E5   20 63 92   JSR $9263    BASIN
90E8   B0 E6      BCS $90D0    Wenn Fehler, Fehlerroutine
90EA   60         RTS

****************************** BASIC-CKOUT

90EB   48         PHA          (A) retten
90EC   20 45 A8   JSR $A845    ROMs einschalten
90EF   20 C9 FF   JSR $FFC9    CKOUT Ausgabegerät setzen
90F2   20 43 92   JSR $9243    DS$ für ungültig erklären
90F5   AA         TAX          Fehlercode in (X)
90F6   68         PLA          (A) wieder holen
90F7   90 03      BCC $90FC    Wenn kein Fehler, Skip
90F9   8A         TXA          Fehlercode wieder in (A)
90FA   B0 D4      BCS $90D0    Wenn Fehler, Fehlerroutine
90FC   60         RTS

****************************** BASIC-CHKIN

90FD   20 45 A8   JSR $A845    ROMs einschalten
9100   20 C6 FF   JSR $FFC6    CHKIN Eingabegerät setzen
9103   20 43 92   JSR $9243    DS$ für ungültig erklären
9106   B0 C8      BCS $90D0    Wenn Fehler, Fehlerroutine
9108   60         RTS

****************************** BASIC-GETIN

9109   20 45 A8   JSR $A845    ROMs einschalten
910C   20 E4 FF   JSR $FFE4    GETIN
910F   B0 C2      BCS $90D3    Wenn Fehler, Fehlerroutine
9111   60         RTS

****************************** BASIC-Befehl SAVE

9112   20 AE 91   JSR $91AE    Parameter holen
9115   AE 10 12   LDX $1210    Zeiger auf Programmende
9118   AC 11 12   LDY $1211    laden
911B   A9 2D      LDA #$2D     Zeiger auf Programmstart in ($2D)
911D   20 45 A8   JSR $A845    ROMs einschalten
9120   20 D8 FF   JSR $FFD8    SAVE
9123   20 43 92   JSR $9243    DS$ für ungültig erklären
9126   B0 A8      BCS $90D0    Wenn Fehler, Fehlerroutine
9128   60         RTS
```

```
****************************** BASIC-Befehl VERIFY

9129   A9 01      LDA #$01     Verify-Flag setzen
912B   2C         .BYTE $2C    Nächsten Befehl überlesen

****************************** BASIC-Befehl LOAD

912C   A9 00      LDA #$00     Load-Flag setzen
912E   85 0C      STA $0C      und speichern
9130   20 AE 91   JSR $91AE    Parameter holen
9133   20 45 A8   JSR $A845    ROMs einschalten
9136   A5 0C      LDA $0C      Load/Verify-Flag
9138   A6 2D      LDX $2D      Zeiger auf Programmstart
913A   A4 2E      LDY $2E      laden
913C   20 D5 FF   JSR $FFD5    LOAD
913F   08         PHP          Status retten
9140   20 43 92   JSR $9243    DS$ für ungültig erklären
9143   28         PLP          Fehler ?
9144   B0 65      BCS $91AB    Ja, zur Fehlerroutine
9146   A5 0C      LDA $0C      LOAD ?
9148   F0 16      BEQ $9160    Ja, Skip
914A   A2 1C      LDX #$1C     Nummer für 'VERIFY ERROR'
914C   20 51 92   JSR $9251    Status holen
914F   29 10      AND #$10     Fehlerbit gesetzt ?
9151   D0 16      BNE $9169    Ja, Fehlerausgabe
9153   24 7F      BIT $7F      Direktmodus ?
9155   30 08      BMI $915F    Nein, Skip
9157   20 81 92   JSR $9281    Text ausgeben

****************************** Textkonstante (CR) 'OK' (CR)

915A   0D 4F 4B 0D 00          (CR) 'OK' (CR)

******************************

915F   60         RTS

9160   20 51 92   JSR $9251    Status holen
9163   29 BF      AND #$BF     Fehlerfrei ?
9165   F0 05      BEQ $916C    Ja, Skip
9167   A2 1D      LDX #$1D     'LOAD'
9169   4C 3C 4D   JMP $4D3C    Fehlerausgabe
916C   8E 10 12   STX $1210    Neues Programmende setzen
916F   8C 11 12   STY $1211
9172   24 7F      BIT $7F      Direktmodus ?
9174   30 0E      BMI $9184    Nein, Skip
9176   70 E7      BVS $915F    Overlay berücksichtigen
9178   20 2A 4D   JSR $4D2A    'READY.' ausgeben
```

```
917B    20 4F 4F    JSR $4F4F    Verkettung korrigieren
917E    20 F3 51    JSR $51F3    PC auf Programmstart, CLR
9181    4C C3 4D    JMP $4DC3    Zurück zum BASIC
9184    20 54 52    JSR $5254    PC auf Programmstart
9187    20 4F 4F    JSR $4F4F    Verkettung korrigieren
918A    4C 35 52    JMP $5235    RESTORE, BASIC initialisieren

****************************** BASIC-Befehl OPEN

918D    20 F6 91    JSR $91F6    Parameter holen
9190    18          CLC
9191    20 D8 90    JSR $90D8    OPEN
9194    20 43 92    JSR $9243    DS$ für ungültig erklären
9197    B0 12       BCS $91AB    Wenn Fehler, Fehlerroutine
9199    60          RTS

****************************** BASIC-Befehl CLOSE

919A    20 F6 91    JSR $91F6    Parameter holen
919D    20 45 A8    JSR $A845    ROMs einschalten
91A0    A5 4B       LDA $4B      Filenummer
91A2    18          CLC
91A3    20 75 92    JSR $9275    CLOSE
91A6    20 43 92    JSR $9243    DS$ für ungültig erklären
91A9    90 B4       BCC $915F    Wenn kein Fehler, Ende
91AB    4C D0 90    JMP $90D0    Fehlerauswertung

****************************** Parameter für LOAD/SAVE holen

91AE    A9 00       LDA #$00     Länge des Filnamens = 0
91B0    20 5D 92    JSR $925D    Filenamenparameter vorsetzen
91B3    A2 01       LDX #$01     Gerätenummer = 1
91B5    A0 00       LDY #$00     Sekundäradresse = 0
91B7    20 57 92    JSR $9257    Fileparameter vorbesetzen
91BA    20 87 92    JSR $9287    Bank setzen
91BD    20 E3 91    JSR $91E3    Auf weitere Zeichen testen
91C0    20 39 92    JSR $9239    Filenamen auswerten
91C3    20 E3 91    JSR $91E3    Auf weitere Zeichen testen
91C6    20 DD 91    JSR $91DD    Gerätenummer holen
91C9    A0 00       LDY #$00     Sekundäradresse auf 0
91CB    86 4B       STX $4B      Geräteadresse setzen
91CD    20 57 92    JSR $9257    Fileparameter setzen
91D0    20 E3 91    JSR $91E3    Auf weiter Zeichen testen
91D3    20 DD 91    JSR $91DD    Sekundäradresse holen
91D6    8A          TXA          (X) in (Y) bringen
91D7    A8          TAY
91D8    A6 4B       LDX $4B      Geräteadresse laden
```

```
91DA   4C 57 92   JMP $9257    Filparameter setzen

****************************** Byte-Wert nach Komma holen

91DD   20 EB 91   JSR $91EB    Auf Komma und weitere Zeichen prüfen
91E0   4C F4 87   JMP $87F4    Byte-Wert holen

****************************** Auf weitere Zeichen testen

91E3   20 86 03   JSR $0386    CHRGOT
91E6   D0 02      BNE $91EA    Wenn gültiges Zeichen, Skip
91E8   68         PLA          Rückkehr zur übergeordneten
91E9   68         PLA          Routine
91EA   60         RTS

****************************** Test auf Komma und weitere Zeichen

91EB   20 5C 79   JSR $795C    Test auf Komma
91EE   20 86 03   JSR $0386    CHRGOT
91F1   D0 F7      BNE $91EA    Wenn gültig, RTS
91F3   4C 6C 79   JMP $796C    'SYNTAX'

****************************** Parameter für OPEN/CLOSE holen

91F6   A9 00      LDA #$00     Länge des Filenamens = 0
91F8   A2 01      LDX #$01     Bank 1 für Dateinamen
91FA   20 87 92   JSR $9287    SETBNK Bank setzen
91FD   20 5D 92   JSR $925D    Filenamenparameter setzen
9200   20 EE 91   JSR $91EE    Auf weitere Zeichen prüfen
9203   20 F4 87   JSR $87F4    Filenummer in (X)
9206   86 4B      STX $4B      Filenummer speichern
9208   8A         TXA          Filenummer in (A)
9209   A2 01      LDX #$01     Geräteadresse auf 1
920B   A0 00      LDY #$00     Sekundäradresse auf 0
920D   20 57 92   JSR $9257    Fileparameter setzen
9210   20 E3 91   JSR $91E3    Auf weitere Zeichen prüfen
9213   20 DD 91   JSR $91DD    Geräteadresse holen
9216   86 4C      STX $4C      und speichern
9218   A0 00      LDY #$00     Sekundäradresse auf 0
921A   A5 4B      LDA $4B      Filenummer
921C   E0 03      CPX #$03     < 3 ?
921E   90 01      BCC $9221    Nein, Skip
9220   88         DEY          Sekundäradresse auf 255
9221   20 57 92   JSR $9257    Fileparameter setzen
9224   20 E3 91   JSR $91E3    Auf weiter Zeichen prüfen
9227   20 DD 91   JSR $91DD    Sekundäradresse holen
922A   8A         TXA          (X) und (Y)
922B   A8         TAY          vertauschen
```

```
922C    A6 4C      LDX $4C      Geräteadresse
922E    A5 4B      LDA $4B      Filenummer
9230    20 57 92   JSR $9257    Fileparameter setzen
9233    20 E3 91   JSR $91E3    Auf weitere Zeichen prüfen
9236    20 EB 91   JSR $91EB    Auf Komma und weitere Zeichen prüfen
9239    20 7B 87   JSR $877B    Stringausdruck auswerten und Parameter
holen
923C    A6 24      LDX $24      Adresse des Filenamens
923E    A4 25      LDY $25      laden
9240    4C 5D 92   JMP $925D    Filenamenparameter setzen

******************************* DS$ falls nötig für ungültig erklären

9243    08         PHP          Status retten
9244    48         PHA          (A) retten
9245    A5 BA      LDA $BA      Geräteadresse
9247    C9 04      CMP #$04     < 4 ?
9249    90 03      BCC $924E    Ja, Skip
924B    20 0D A8   JSR $A80D    DS$ für ungültig erklären
924E    68         PLA          (A) wieder laden
924F    28         PLP          Status laden
9250    60         RTS

******************************* BASIC-Status holen

9251    20 45 A8   JSR $A845    ROMs einschalten
9254    4C B7 FF   JMP $FFB7    READST Status lesen

******************************* BASIC-Fileparameter setzen

9257    20 45 A8   JSR $A845    ROMs einschalten
925A    4C BA FF   JMP $FFBA    SETLFS Fileparameter setzen

******************************* BASIC-Filenamenparameter setzen

925D    20 45 A8   JSR $A845    ROMs einschalten
9260    4C BD FF   JMP $FFBD    SETNAM Filenamenparameter setzen

******************************* BASIC-BASIN (entspricht INPUT)

9263    20 45 A8   JSR $A845    ROMs einschalten
9266    4C CF FF   JMP $FFCF    BASIN

******************************* BASIC-BSOUT Zeichenausgabe

9269    20 45 A8   JSR $A845    ROMs einschalten
926C    4C D2 FF   JMP $FFD2    BSOUT Zeichenausgabe
```

```
******************************* BASIC-CLRCH I/O rücksetzen

926F   20 45 A8   JSR $A845    ROMs einschalten
9272   4C CC FF   JMP $FFCC    CLRCH

******************************* BASIC-CLOSE

9275   20 45 A8   JSR $A845    ROMs einschalten
9278   4C C3 FF   JMP $FFC3    CLOSE

******************************* BASIC-CLALL

927B   20 45 A8   JSR $A845    ROMs einschalten
927E   4C E7 FF   JMP $FFE7    CLALL

******************************* BASIC-PRIMM Textausgabe

9281   20 45 A8   JSR $A845    ROMs einschalten
9284   4C 7D FF   JMP $FF7D    PRIMM

******************************* BASIC-SETBNK Bank für I/O-Operationen
setzen

9287   20 45 A8   JSR $A845    ROMs einschalten
928A   4C 68 FF   JMP $FF68    SETBNK

******************************* BASIC-PLOT Cursor setzen

928D   8D 03 FF   STA $FF03    Write in Bank 0 setzen
9290   4C F0 FF   JMP $FFF0    PLOT

******************************* BASIC-STOP Test auf STOP-Taste

9293   20 45 A8   JSR $A845    ROMs einschalten
9296   4C E1 FF   JMP $FFE1    STOP

******************************* Platz für String in Stringbereich
                                reservieren, Länge in (A)

9299   46 11      LSR $11      Garbage-Collection-Flag löschen
929B   AA         TAX          Länge = 0 ?
929C   F0 3B      BEQ $92D9    Ja, Ende
929E   48         PHA          Stringlänge retten
929F   A5 35      LDA $35      Stringspeicherstart
92A1   38         SEC
92A2   E9 02      SBC #$02     - 2 Bytes für Deskriptorpointer
92A4   A4 36      LDY $36
92A6   B0 01      BCS $92A9    Kein Übertrag, Skip
```

```
92A8   88        DEY            Übertrag berücksichtigen
92A9   85 24     STA $24        = Ende des neuen Strings
92AB   84 25     STY $25
92AD   8A        TXA            Stringlänge wieder laden
92AE   49 FF     EOR #$FF       Wert negieren
92B0   38        SEC            durch Zweierkomplementbildung
92B1   65 24     ADC $24        von Stringlänge subtrahieren
92B3   B0 01     BCS $92B6      Kein Übertrag, Skip
92B5   88        DEY            High-Byte anpassen
92B6   C4 34     CPY $34        Kleiner als Variablenspeicherende ?
92B8   90 20     BCC $92DA      Ja, Garbage-Collection
92BA   D0 04     BNE $92C0      Nein, Skip
92BC   C5 33     CMP $33        Kleiner als variablenspeicherende ?
92BE   90 1A     BCC $92DA      Ja, Garbage-Collection
92C0   85 37     STA $37        Start des neuen Strings
92C2   84 38     STY $38        setzen
92C4   A0 01     LDY #$01       Offset auf zweites Trailerbyte
92C6   A9 FF     LDA #$FF       Trailerwert für ungültigen String
92C8   8D 04 FF  STA $FF04      Write in Bank 1 setzen
92CB   91 24     STA ($24),Y    Trailer auf ungültig setzen
92CD   88        DEY            Offset auf erstes Trailerbyte setzen
92CE   68        PLA            Stringlänge wieder holen
92CF   91 24     STA ($24),Y    und Trailer setzen
92D1   A6 37     LDX $37        Adresse des neuen Strings
92D3   A4 38     LDY $38        laden
92D5   86 35     STX $35        Neuen Stringspeicherstart
92D7   84 36     STY $36        setzen
92D9   60        RTS

92DA   A5 11     LDA $11        Garbage-Collection schon durchgeführt ?
92DC   30 09     BMI $92E7      Ja, Fehler
92DE   20 EA 92  JSR $92EA      Garbage-Collection ausführen
92E1   38        SEC            Bit 7 im
92E2   66 11     ROR $11        Flag setzen
92E4   68        PLA
92E5   D0 B4     BNE $929B      Und zurück zum Aufruf
92E7   4C 3A 4D  JMP $4D3A      'OUT OF MEMORY'

****************************** Garbage-Collection

92EA   A6 18     LDX $18        Stringstackpointer laden
92EC   E0 1B     CPX #$1B       Stringstack leer ?
92EE   F0 13     BEQ $9303      Ja, Skip
92F0   20 F0 93  JSR $93F0      Deskriptorwerte aus Stringstack holen
92F3   F0 F7     BEQ $92EC      Wenn Länge = 0, weitertesten
92F5   8A        TXA            Stringstackpointer in (A)
92F6   A0 00     LDY #$00       Offset auf 0 setzen
92F8   8D 04 FF  STA $FF04      Write in Bank 1 setzen
```

```
92FB   91 5E      STA ($5E),Y  Trailer auf Stringstack setzen
92FD   98         TYA          (A)=0
92FE   C8         INY          Offset erhöhen
92FF   91 5E      STA ($5E),Y  High-Byte des Trailers auf 0 setzen
9301   D0 E9      BNE $92EC    Weitertesten

9303   A0 00      LDY #$00
9305   84 5A      STY $5A      Flag für ungültigen String löschen
9307   A6 39      LDX $39      Stringspeicherendezeiger
9309   A4 3A      LDY $3A      laden und
930B   86 61      STX $61      als Zielzeiger  ($61)
930D   86 50      STX $50      als Quellzeiger ($50)
930F   86 37      STX $37      und Movezeiger  ($37)
9311   84 62      STY $62      setzen
9313   84 51      STY $51
9315   84 38      STY $38
9317   8A         TXA          High-Byte der Adresse in (A)
9318   20 83 93   JSR $9383    Nächsten String testen
931B   D0 0C      BNE $9329    Wenn String gültig, Skip
931D   88         DEY          Offset erniedrigen
931E   20 FB 42   JSR $42FB    1. Trailerbyte (Stringlänge) laden
9321   20 D2 93   JSR $93D2    Länge von Quellzeiger abziehen
9324   38         SEC          Bit 7 im Flag
9325   66 5A      ROR $5A      setzen
9327   D0 EF      BNE $9318    Weiter testen

9329   24 5A      BIT $5A      Ungültiger String gefunden ?
932B   10 42      BPL $936F    Nein, Skip
932D   A2 00      LDX #$00     Flag wieder löschen
932F   86 5A      STX $5A
9331   A9 02      LDA #$02     ???
9333   A0 01      LDY #$01     Trailer an Zielposition verschieben
9335   20 FB 42   JSR $42FB    LDA ($50),Y Bank 1
9338   91 61      STA ($61),Y  Trailerbyte setzen
933A   88         DEY          Offset erniedrigen
933B   20 FB 42   JSR $42FB    Trailerbyte laden
933E   91 61      STA ($61),Y  und setzen
9340   20 B7 03   JSR $03B7    Länge aus Deskriptor laden
9343   AA         TAX          und retten
9344   20 E1 93   JSR $93E1    Länge von Zielzeiger abziehen
9347   85 37      STA $37      und als Movepointer setzen
9349   84 38      STY $38
934B   8A         TXA          Länge wieder holen
934C   20 D2 93   JSR $93D2    und von Quellzeiger abziehen
934F   8A         TXA          Länge in (Y) bringen
9350   A8         TAY
9351   88         DEY          String byteweise übertragen
9352   20 FB 42   JSR $42FB    Stringbyte laden
```

```
9355   91 61      STA ($61),Y  und kopieren
9357   CA         DEX          Alle Bytes kopiert ?
9358   D0 F7      BNE $9351    Nein, nochmal
935A   A0 02      LDY #$02     Offset laden
935C   B9 60 00   LDA $0060,Y  Deskriptor laden
935F   91 24      STA ($24),Y  und neuen Deskriptor setzen
9361   88         DEY          Deskriptor kopiert ?
9362   D0 F8      BNE $935C    Nein, nochmal
9364   A5 50      LDA $50      Quellzeiger laden
9366   A4 51      LDY $51
9368   20 83 93   JSR $9383    Ungültiger String erreicht ?
936B   F0 B0      BEQ $931D    Ja, Zeiger korrigieren
936D   D0 C4      BNE $9333    Nein, weiterkopieren

936F   A0 00      LDY #$00     Offset auf 0 setzen
9371   20 B7 03   JSR $03B7    Stringlänge holen
9374   AA         TAX          und in (X) retten
9375   20 E1 93   JSR $93E1    Von Zielzeiger abziehen
9378   85 37      STA $37      und als neue Untergrenze
937A   84 38      STY $38      setzen
937C   8A         TXA          Länge
937D   20 D2 93   JSR $93D2    von Quellzeiger abziehen
9380   4C 18 93   JMP $9318    und weiterschleifen

****************************** String übernehmen falls möglich

9383   C4 36      CPY $36      (A)/(Y) < Stringspeicherstart ?
9385   90 2A      BCC $93B1    Ja, Stringstack testen
9387   D0 06      BNE $938F    Nein, Skip
9389   C5 35      CMP $35      (A)/(Y) < Stringspeicherstart ?
938B   F0 24      BEQ $93B1    Ja, Stringstack testen
938D   90 22      BCC $93B1    Ja, Stringstack testen
938F   24 5A      BIT $5A      Flag gesetzt ?
9391   30 05      BMI $9398    Ja, Skip
9393   A9 02      LDA #$02
9395   20 E1 93   JSR $93E1    Zielzeiger um 2 erniedrigen
9398   A9 02      LDA #$02
939A   20 D2 93   JSR $93D2    Quellzeiger um 2 erniedrigen
939D   A0 01      LDY #$01     Offset laden
939F   20 FB 42   JSR $42FB    2. Trailerbyte laden
93A2   C9 FF      CMP #$FF     String ungültig ?
93A4   D0 01      BNE $93A7    Nein, Skip
93A6   60         RTS
93A7   20 FB 42   JSR $42FB    Trailerbyte laden
93AA   99 24 00   STA $0024,Y  und in Hilfszeiger schreiben
93AD   88         DEY          Trailer kopiert ?
93AE   10 F7      BPL $93A7    Nein, nochmal
```

```
93B0   60         RTS

93B1   A6 18      LDX $18       Stringstackpointer laden
93B3   E0 1B      CPX #$1B      Stringstack leer ?
93B5   F0 10      BEQ $93C7     Ja, Ende
93B7   20 F0 93   JSR $93F0     Deskriptordaten aus Stringstack holen
93BA   F0 F7      BEQ $93B3     Wenn Länge = 0, weitertesten
93BC   A0 00      LDY #$00      Offset auf 0 setzen
93BE   91 5E      STA ($5E),Y   1. Trailerbyte auf 0
93C0   C8         INY           Offset erhöhen
93C1   A9 FF      LDA #$FF      2. Trailerbyte auf $FF
93C3   91 5E      STA ($5E),Y   = String ungültig
93C5   D0 EC      BNE $93B3     Weiter testen

93C7   68         PLA           Rücksprungadresse vom Stapel
93C8   68         PLA           löschen
93C9   A5 37      LDA $37       Hilfszeiger laden
93CB   A4 38      LDY $38       und
93CD   85 35      STA $35       Stringspeicherstart neu
93CF   84 36      STY $36       setzen
93D1   60         RTS

****************************** Quellzeiger um (A) erniedrigen

93D2   49 FF      EOR #$FF      Zweierkomplement bilden
93D4   38         SEC
93D5   65 50      ADC $50       und damit (A) abziehen
93D7   A4 51      LDY $51       High-Byte laden
93D9   B0 01      BCS $93DC     Kein Übertrag, Skip
93DB   88         DEY           Übertrag berücksichtigen
93DC   85 50      STA $50       Quellzeiger neu setzen
93DE   84 51      STY $51
93E0   60         RTS

****************************** Zielzeiger um (A) erniedrigen

93E1   49 FF      EOR #$FF      Zweierkomplement
93E3   38         SEC
93E4   65 61      ADC $61       und damit (A) abziehen
93E6   A4 62      LDY $62       High-Byte laden
93E8   B0 01      BCS $93EB     Kein Übertrag, Skip
93EA   88         DEY           Übertrag berücksichtigen
93EB   85 61      STA $61       Zielzeiger neu setzen
93ED   84 62      STY $62
93EF   60         RTS
```

```
******************************* Deskriptorwerte aus Stringstack holen

93F0   CA         DEX          Offset erniedrigen
93F1   B5 00      LDA $00,X    Adresse High setzen
93F3   85 5F      STA $5F
93F5   CA         DEX          Offset erniedrigen
93F6   B5 00      LDA $00,X    Adresse Low setzen
93F8   85 5E      STA $5E
93FA   CA         DEX          Offset erniedrigen
93FB   B5 00      LDA $00,X    Länge holen
93FD   48         PHA          und retten
93FE   18         CLC
93FF   65 5E      ADC $5E      Länge auf Hilfszeiger
9401   85 5E      STA $5E      aufaddieren
9403   90 02      BCC $9407    Kein Übertrag, Skip
9405   E6 5F      INC $5F      Übetrag berücksichtigen
9407   68         PLA          Länge holen
9408   60         RTS

******************************* BASIC-Funktion COS

9409   A9 85      LDA #$85     Zeiger auf Konstante PI/2
940B   A0 94      LDY #$94     ab $9485 setzen
940D   20 12 8A   JSR $8A12    FAC#1 = Konstante + FAC#1

******************************* BASIC-Funktion SIN

9410   20 38 8C   JSR $8C38    FAC#2 = FAC#1 mit Rundung
9413   A9 8A      LDA #$8A     Zeiger auf Konstante PI*2
9415   A0 94      LDY #$94     ab $948A setzen
9417   A6 6F      LDX $6F      FAC#2 Vorzeichen
9419   20 41 8B   JSR $8B41    FAC#1 = FAC#1 / Konstante
941C   20 38 8C   JSR $8C38    FAC#2 = FAC#1 mit Rundung
941F   20 FB 8C   JSR $8CFB    INT-Funktion
9422   A9 00      LDA #$00
9424   85 70      STA $70
9426   20 31 88   JSR $8831    FAC#1 = FAC#2 - FAC#1
9429   A9 8F      LDA #$8F     Zeiger auf Konstante 0.25
942B   A0 94      LDY #$94     ab $948F setzen
942D   20 18 8A   JSR $8A18    FAC#1 = Konstante - FAC#1
9430   A5 68      LDA $68      FAC#1 Vorzeichen
9432   48         PHA          auf Stack
9433   10 0D      BPL $9442    Wenn positiv, Skip
9435   20 0E 8A   JSR $8A0E    FAC#1 = FAC#1 + 0.5
9438   A5 68      LDA $68      FAC#1 negativ ?
943A   30 09      BMI $9445    Ja, Skip
943C   A5 14      LDA $14
943E   49 FF      EOR #$FF     Flag invertieren
```

```
9440   85 14      STA $14
9442   20 FA 8F   JSR $8FFA   FAC#1 Vorzeichenwechsel
9445   A9 8F      LDA #$8F    Zeiger auf Konstante 0.25
9447   A0 94      LDY #$94    ab $948F setzen
9449   20 12 8A   JSR $8A12   FAC#1 = FAC#1 + 0.25
944C   68         PLA         Altes Vorzeichen holen
944D   10 03      BPL $9452   Wenn positiv, Skip
944F   20 FA 8F   JSR $8FFA   FAC#1 Vorzeichenwechsel
9452   A9 94      LDA #$94    Zeiger auf Polynomkoeffizienten
9454   A0 94      LDY #$94    ab $9494 setzen
9456   4C 86 90   JMP $9086   Zur Polynomberechnung

****************************** BASIC-Funktion TAN

9459   20 FC 8B   JSR $8BFC   FAC#1 in Akku 3
945C   A9 00      LDA #$00    Flag löschen
945E   85 14      STA $14
9460   20 10 94   JSR $9410   SIN-Funktion
9463   A2 50      LDX #$50    Zeiger auf Hilfsakku
9465   A0 00      LDY #$00    ab $0050 setzen
9467   20 00 8C   JSR $8C00   FAC#1 in Hilfsakku mit Rundung
946A   A9 59      LDA #$59    Zeiger auf Akku 3
946C   A0 00      LDY #$00    ab $0059 setzen
946E   20 D4 8B   JSR $8BD4   FAC#1 = Konstante
9471   A9 00      LDA #$00    FAC#1 Vorzeichen löschen
9473   85 68      STA $68
9475   A5 14      LDA $14     Flag holen
9477   20 81 94   JSR $9481   COS berechnen
947A   A9 50      LDA #$50    Zeiger auf Hilfsakku
947C   A0 00      LDY #$00    ab $0050 setzen
947E   4C 49 8B   JMP $8B49   FAC#1 = Konstante / FAC#1
9481   48         PHA         Flag auf Stack
9482   4C 42 94   JMP $9442   Zur COS-Funktion

****************************** Konstanten für SIN/COS/TAN

9485   81 49 0F DA A2      1.57079633   PI/2
948A   83 49 0F DA A2      6.28318531   PI*2
948F   7F 00 00 00 00      0.25
9494   05                 5 = Polynomgrad, 6 Koeffizienten
9495   84 E6 1A 2D 1B     -14.3813907
949A   86 28 07 FB F8      42.0077971
949F   87 99 68 89 01     -76.7041703
94A4   87 23 35 DF E1      81.6052237
94A9   86 A5 5D E7 28     -41.3147021
94AE   83 49 0F DA A2      6.28318531   PI*2
```

```
******************************* BASIC-Funktion ATN

94B3   A5 68      LDA $68      FAC#1 Vorzeichen retten
94B5   48         PHA
94B6   10 03      BPL $94BB    Wenn positiv, Skip
94B8   20 FA 8F   JSR $8FFA    FAC#1 Vorzeichenwechsel
94BB   A5 63      LDA $63      FAC#1 Exponent retten
94BD   48         PHA
94BE   C9 81      CMP #$81     Vergleich mit 1
94C0   90 07      BCC $94C9    Wenn kleiner, Skip
94C2   A9 9C      LDA #$9C     Zeiger auf Konstante 1
94C4   A0 89      LDY #$89     ab $899C setzen
94C6   20 1E 8A   JSR $8A1E    FAC#1 durch 1 dividieren = Kehrwert
94C9   A9 E3      LDA #$E3     Zeiger auf Polynomkoeffizienten
94CB   A0 94      LDY #$94     ab $94E3 setzen
94CD   20 86 90   JSR $9086    Zur Polynomberechnung
94D0   68         PLA          Exponent wieder holen
94D1   C9 81      CMP #$81     Alte Zahl < 1 ?
94D3   90 07      BCC $94DC    Ja, Skip
94D5   A9 85      LDA #$85     Zeiger auf Konstante PI/2
94D7   A0 94      LDY #$94     ab $9485 setzen
94D9   20 18 8A   JSR $8A18    FAC#1 = Konstante - FAC#1
94DC   68         PLA          Altes Vorzeichen holen
94DD   10 03      BPL $94E2    Wenn positiv, Ende
94DF   4C FA 8F   JMP $8FFA    FAC#1 Vorzeichenwechsel
94E2   60         RTS

******************************* Konstanten für ATN

94E3   0B                      11 = Polynomgrad, 12 Koeffizienten
94E4   76 B3 83 BD D3          -6.84793912E-04
94E9   79 1E F4 A6 F5           4.85094216E-03
94EE   7B 83 FC B0 10          -.0161117015
94F3   7C 0C 1F 67 CA           .034209638
94F8   7C DE 53 CB C1          -.054279133
94FD   7D 14 64 70 4C           .0724571965
9502   7D B7 EA 51 7A          -.0898019185
9507   7D 63 30 88 7E           .110932413
950C   7E 92 44 99 3A          -.142839808
9511   7E 4C CC 91 C7           .19999912
9516   7F AA AA AA 13          -.333333316
951B   81 00 00 00 00           1

******************************* BASIC-Befehl PRINT USING

9520   A2 FF      LDX #$FF     Zeiger auf Feldbeginn löschen
9522   8E 36 01   STX $0136
9525   20 80 03   JSR $0380    CHRGET
```

```
9528   20 EF 77   JSR $77EF    FRMEVL Ausdruck auswerten
952B   20 DD 77   JSR $77DD    Auf String testen
952E   A5 66      LDA $66      Deskriptoradresse retten
9530   48         PHA
9531   A5 67      LDA $67
9533   48         PHA
9534   A0 02      LDY #$02     Offset laden
9536   20 E7 42   JSR $42E7    Deskriptor in Hilfszeiger übertragen
9539   88         DEY          Offset erniedrigen
953A   99 3F 00   STA $003F,Y  Hilfszeiger setzen
953D   D0 F7      BNE $9536    Offset noch nicht 0, nochmal
953F   20 E7 42   JSR $42E7    Länge des Formatstrings holen
9542   8D 35 01   STA $0135    und setzen
9545   A8         TAY          Länge = 0 ?
9546   F0 0B      BEQ $9553    Ja, Fehler
9548   88         DEY          Länge als Offset erniedrigen
9549   20 D3 42   JSR $42D3    Zeichen aus String holen
954C   C9 23      CMP #$23     Ein '#' vorhanden ?
954E   F0 06      BEQ $9556    Ja, Skip
9550   98         TYA          Stringanfang erreicht ?
9551   D0 F5      BNE $9548    Nein, weitersuchen
9553   4C 6C 79   JMP $796C    'SYNTAX'
9556   A9 3B      LDA #$3B     ';'
9558   20 5E 79   JSR $795E    Prüft auf Code
955B   84 77      STY $77      Ausgabeflag löschen, (Y) = 0 !
955D   8C 23 01   STY $0123    Anfangszeiger auf 0
9560   20 EF 77   JSR $77EF    FRMEVL Ausdruck auswerten
9563   24 0F      BIT $0F      Numerisch ?
9565   10 39      BPL $95A0    Ja, Skip
9567   20 9F 97   JSR $979F    Zeiger vorbesetzen
956A   20 F2 98   JSR $98F2    Zeichenausgabe und Syntaxprüfung
956D   AE 2B 01   LDX $012B    Flag für Ausrichtung gesetzt ?
9570   F0 15      BEQ $9587    Nein, Skip
9572   A2 00      LDX #$00     Ausrichtung mit 0 vorbesetzen
9574   38         SEC
9575   AD 31 01   LDA $0131    Feldlänge
9578   E5 78      SBC $78      - Stringlänge
957A   90 0B      BCC $9587    Wenn kein Platz zum Ausrichten, Skip
957C   A2 3D      LDX #$3D     '=' ? (Zentrieren)
957E   EC 2B 01   CPX $012B
9581   D0 03      BNE $9586    Nein, Skip
9583   4A         LSR          Ausrichtungwert halbieren
9584   69 00      ADC #$00     Und ungeraden Wert berücksichtigen
9586   AA         TAX          Ausrichtungswert setzen
9587   A0 00      LDY #$00     Offset auf Start
9589   8A         TXA          Ausrichtung ?
958A   F0 05      BEQ $9591    Nein, Skip
958C   CA         DEX          Zähler dekrementieren
```

```
958D   A9 20      LDA #$20      und Space
958F   D0 08      BNE $9599     ausgeben

9591   C4 78      CPY $78       Stringende erreicht ?
9593   B0 F8      BCS $958D     Ja, Space ausgeben
9595   20 B7 03   JSR $03B7     Zeichen aus String holen
9598   C8         INY           Offset erhöhen
9599   20 EB 98   JSR $98EB     Zeichen auf Schirm ausgeben
959C   D0 EB      BNE $9589     Solange kein Ende, weitermachen
959E   F0 27      BEQ $95C7     Unbedingter Sprung

95A0   20 42 8E   JSR $8E42     FAC#1 in ASCII wandeln
95A3   A0 FF      LDY #$FF      Offset auf Start
95A5   C8         INY           Offset erhöhen
95A6   B9 00 01   LDA $0100,Y   Ende erreicht ?
95A9   D0 FA      BNE $95A5     Nein, weitersuchen
95AB   98         TYA           Länge in (A)
95AC   20 90 86   JSR $8690     Platz reservieren und Zeiger setzen
95AF   A0 00      LDY #$00      Offset auf 0 setzen
95B1   8D 04 FF   STA $FF04     Write in Bank 1 setzen
95B4   B9 00 01   LDA $0100,Y   String in reservierten Bereich übertragen
95B7   F0 05      BEQ $95BE     Wenn Ende, Skip
95B9   91 64      STA ($64),Y   String übertragen
95BB   C8         INY           Offset erhöhen
95BC   D0 F6      BNE $95B4     Unbedingter Sprung
95BE   20 E3 86   JSR $86E3     Stringzeiger in Stringstack bringen
95C1   20 9F 97   JSR $979F     Zeiger vorbesetzen
95C4   20 E7 95   JSR $95E7     Zahl formatieren

95C7   20 86 03   JSR $0386     CHRGOT
95CA   C9 2C      CMP #$2C      = ','
95CC   F0 8A      BEQ $9558     Ja, zum Schleifenstart
95CE   38         SEC           Ausgabeflag setzen
95CF   66 77      ROR $77
95D1   20 F2 98   JSR $98F2     String formatiert ausgeben
95D4   68         PLA           Deskriptoradresse wieder holen
95D5   A8         TAY           High-Byte in (Y)
95D6   68         PLA           Low-Byte holen
95D7   20 85 87   JSR $8785     FRESTR
95DA   20 86 03   JSR $0386     CHRGOT
95DD   C9 3B      CMP #$3B      = ';' ?
95DF   F0 03      BEQ $95E4     Ja, Skip
95E1   4C 98 55   JMP $5598     CR out
95E4   4C 80 03   JMP $0380     CHRGET, Ende

****************************** Zahl formatiert ausgeben

95E7   8D 03 FF   STA $FF03     Read aus Bank 0 setzen
```

```
95EA  AD 04 12   LDA $1204     Füllzeichen
95ED  8D 33 01   STA $0133     vorbesetzen
95F0  A9 FF      LDA #$FF      Vorzeichenflag löschen
95F2  8D 32 01   STA $0132
95F5  4C FA 95   JMP $95FA

95F8  86 80      STX $80       Dezimalpunktstelle setzen

95FA  C4 78      CPY $78       Ende erreicht ?
95FC  F0 33      BEQ $9631     Ja, Skip
95FE  B9 00 01   LDA $0100,Y   Zeichen aus Puffer holen
9601  C8         INY           Pufferzeiger erhöhen
9602  C9 20      CMP #$20      ' ' ?
9604  F0 F4      BEQ $95FA     Ja, weiter suchen
9606  C9 2D      CMP #$2D      '-' ?
9608  F0 E8      BEQ $95F2     Ja, Vorzeichen setzen
960A  C9 2E      CMP #$2E      '.' ?
960C  F0 EA      BEQ $95F8     Ja, Dezimalpunktstelle setzen
960E  C9 45      CMP #$45      'E' ?
9610  F0 11      BEQ $9623     Ja, Exponent setzen
9612  9D 00 01   STA $0100,X   Zeichen wieder in Puffer
9615  8E 24 01   STX $0124     Endezeiger setzen
9618  E8         INX           Offset erhöhen
9619  24 80      BIT $80       Dezimalpunktstelle schon gesetzt ?
961B  10 DD      BPL $95FA     Ja, weiter testen
961D  EE 2A 01   INC $012A     Zähler für Vorkommastellen erhöhen
9620  4C FA 95   JMP $95FA     und weitertesten

9623  B9 00 01   LDA $0100,Y   Nächstes Zeichen
9626  C9 2D      CMP #$2D      = '-' ?
9628  D0 03      BNE $962D     Nein, Skip
962A  6E 28 01   ROR $0128     E-Negativflag setzen
962D  C8         INY           Offset erhöhen
962E  8C 29 01   STY $0129     Zeiger auf Exponenten setzen
9631  A5 80      LDA $80       Dezimalpunktstelle schon gesetzt ?
9633  10 02      BPL $9637     Ja, Skip
9635  86 80      STX $80       Dezimalpunktstelle setzen
9637  20 F2 98   JSR $98F2     Zeichenausgabe und Syntaxprüfung
963A  AD 2C 01   LDA $012C     Platz für '$' ?
963D  C9 FF      CMP #$FF
963F  F0 29      BEQ $966A     Nein, Überlauf
9641  AD 2F 01   LDA $012F     Exponent vorhanden ?
9644  F0 3F      BEQ $9685     Nein, Skip
9646  AD 29 01   LDA $0129     Exponentzeiger gesetzt ?
9649  D0 12      BNE $965D     Ja, Skip
964B  AE 24 01   LDX $0124     Zeiger auf Nummerende laden
964E  20 74 97   JSR $9774     Exponent setzen
9651  DE 02 01   DEC $0102,X   Wert erniedrigen
```

```
9654  E8        INX         Offset erhöhen
9655  8E 29 01  STX $0129   Zeiger auf Exponenten neu setzen
9658  20 FB 97  JSR $97FB   Zahl anpassen
965B  F0 25     BEQ $9682   Fertig, Skip
965D  AC 2E 01  LDY $012E   Vorzeichenflag gesetzt ?
9660  D0 17     BNE $9679   Nein, Skip
9662  AC 32 01  LDY $0132   Vorzeichen gesetzt ?
9665  30 12     BMI $9679   Nein, Skip
9667  AD 2C 01  LDA $012C   Vorkommastellen vorhanden ?
966A  F0 6A     BEQ $96D6   Nein, Überlauf
966C  CE 2C 01  DEC $012C   Wert anpassen
966F  D0 05     BNE $9676   Wenn ausreichend, Skip
9671  AD 2D 01  LDA $012D   Feldpositionen vorhanden ?
9674  F0 60     BEQ $96D6   Nein, Überlauf
9676  EE 27 01  INC $0127   Zähler anpassen
9679  20 EE 96  JSR $96EE   Stellenzahl anpassen
967C  20 B9 97  JSR $97B9   Zahl runden
967F  20 EE 96  JSR $96EE   Stellenzahl anpassen
9682  4C 1C 98  JMP $981C   Zahl ausgeben

9685  AC 29 01  LDY $0129   Exponentzeiger gesetzt ?
9688  F0 16     BEQ $96A0   Nein, Skip
968A  85 78     STA $78     Zähler setzen
968C  38        SEC
968D  6E 30 01  ROR $0130   Flag setzen
9690  A4 80     LDY $80     Zeiger auf Komma laden
9692  AD 28 01  LDA $0128   Vorzeichenexponent positiv ?
9695  10 06     BPL $969D   Ja, Skip
9697  20 27 97  JSR $9727   Stellenzahl anpassen
969A  4C A9 96  JMP $96A9
969D  20 08 97  JSR $9708   Stellenzahl anpassen
96A0  A4 80     LDY $80     Dezimalpunkt an erster Stelle ?
96A2  F0 05     BEQ $96A9   Ja, Skip
96A4  20 FF 97  JSR $97FF   Zahl anpassen
96A7  F0 06     BEQ $96AF   Fertig, Skip
96A9  20 B9 97  JSR $97B9   Zahl runden
96AC  4C B2 96  JMP $96B2
96AF  CE 2A 01  DEC $012A   Vorkommastellenzähler korrigieren
96B2  38        SEC
96B3  AD 2C 01  LDA $012C   Zuwenig Feldpositionen
96B6  ED 2A 01  SBC $012A   für Vorkommastellen ?
96B9  90 1B     BCC $96D6   Ja, Überlauf
96BB  8D 27 01  STA $0127   Restlichen Platz merken
96BE  AC 2E 01  LDY $012E   Vorzeichenflag gesetzt ?
96C1  D0 1B     BNE $96DE   Nein,Skip
96C3  AC 32 01  LDY $0132   Vorzeichen gesetzt ?
96C6  30 16     BMI $96DE   Nein, Skip
96C8  A8        TAY         Platz für Vorzeichen vorhanden ?
```

```
96C9   F0 0B      BEQ $96D6   Nein, Überlauf
96CB   88         DEY         Restplatz vorhanden ?
96CC   D0 13      BNE $96E1   Ja, Skip
96CE   AD 2D 01   LDA $012D   Möglichkeiten ausreichend ?
96D1   0D 2A 01   ORA $012A
96D4   D0 AC      BNE $9682   Ja, Zahl ausgeben
96D6   A9 2A      LDA #$2A    '*'
96D8   20 EB 98   JSR $98EB   ausgeben
96DB   D0 FB      BNE $96D8   Wenn Ende noch nicht erreicht, mehr '*'
96DD   60         RTS

96DE   A8         TAY         Restplatz vorhanden ?
96DF   F0 A1      BEQ $9682   Nein, Zahl ausgeben
96E1   AD 2A 01   LDA $012A   Vorkommastellen vorhanden ?
96E4   D0 9C      BNE $9682   Ja, Zahl ausgeben
96E6   CE 27 01   DEC $0127   Restplatz korrigieren
96E9   E6 77      INC $77     und Zähler erhöhen
96EB   4C 82 96   JMP $9682   Zahl ausgeben

******************************* Stellenzahl anpassen

96EE   38         SEC
96EF   AD 2C 01   LDA $012C   Feldpositionen für Vorkommastellen
96F2   ED 2A 01   SBC $012A   = Vorkommastellenzahl ?
96F5   F0 39      BEQ $9730   Ja, Ende
96F7   A0 80      LDY $80     Zeiger auf Dezimalpunkt laden
96F9   90 16      BCC $9711   Wenn zuwenig Stellen, Skip
96FB    85 78        STA $78        Anzahl der überflüssigen Positionen
setzen
96FD   CC 24 01   CPY $0124   Nummerende erreicht ?
9700   F0 02      BEQ $9704   Ja, Skip
9702   B0 01      BCS $9705   Wenn größer, Skip
9704   C8         INY         Kommaposition korrigieren
9705   EE 2A 01   INC $012A   Vorkommastellen erhöhen
9708   20 3D 97   JSR $973D   Exponent berechnen
970B   C6 78      DEC $78     Alle Stellen aufgearbeitet ?
970D   D0 EE      BNE $96FD   Nein, weitermachen
970F   F0 1D      BEQ $972E   Unbedingter Sprung

9711   49 FF      EOR #$FF    Stellenmangel positiv machen
9713   69 01      ADC #$01
9715   85 78      STA $78     und setzen
9717   CC 23 01   CPY $0123   Anfang erreicht ?
971A   F0 07      BEQ $9723   Ja, Skip
971C   88         DEY         Kommastelle korrigieren
971D   CE 2A 01   DEC $012A   Vorkommastellen erniedrigen
9720   4C 25 97   JMP $9725
9723   E6 77      INC $77     Zähler erhohen
```

```
9725   A9 80      LDA #$80
9727   20 3F 97   JSR $973F    Exponent berechnen
972A   C6 78      DEC $78      Alle fehlenden Stellen aufgearbeitet ?
972C   D0 E9      BNE $9717    Nein, weitermachen
972E   84 80      STY $80      Neue Kommaposition setzen
9730   60         RTS

****************************** Exponentberechnung

9731   D0 39      BNE $976C    Wenn nicht '0' oder '9', Skip
9733   49 09      EOR #$09     Ziffern '0' und '9' vertauschen um
9735   9D 00 01   STA $0100,X  Übertrag auf nächsten Zehner zu setzen
9738   CA         DEX          Offset korrigieren
9739   EC 29 01   CPX $0129
973C   60         RTS

973D   A9 00      LDA #$00     Flag vorbesetzen
973F   AE 29 01   LDX $0129    Zeiger auf Exponenten laden
9742   E8         INX          und auf zweites Zeichen setzen
9743   2C 30 01   BIT $0130    Exponent erniedrigen ?
9746   30 10      BMI $9758    Ja, Skip
9748   4D 28 01   EOR $0128    Vorzeichen = Flag ?
974B   F0 0B      BEQ $9758    Ja, erniedrigen
974D   20 82 97   JSR $9782    Ziffer erhöhen
9750   20 31 97   JSR $9731    Überträge berücksichtigen, Endetest
9753   B0 F8      BCS $974D    Falls nötig, weitermachen
9755   4C 5D 89   JMP $895D    'OVERFLOW'

9758   BD 00 01   LDA $0100,X  Ziffer erniedrigen
975B   DE 00 01   DEC $0100,X
975E   C9 30      CMP #$30     '0' erreicht ?
9760   20 31 97   JSR $9731    Überträge berücksichtigen, Endetest
9763   B0 F3      BCS $9758    Falls nötig, weitermachen
9765   2C 30 01   BIT $0130    Schalter gesetzt ?
9768   10 05      BPL $976F    Nein, Ende
976A   84 80      STY $80      Kommaposition setzen

976C   68         PLA          Rückkehr zur übergeordneten Routine
976D   68         PLA          durch Löschen der Rücksprungadresse
976E   60         RTS

976F   AD 28 01   LDA $0128    Vorzeichen invertieren
9772   49 80      EOR #$80
9774   8D 28 01   STA $0128
9777   A9 30      LDA #$30     '0' Exponentstelle 1
9779   9D 01 01   STA $0101,X  Exponentstelle setzen
977C   A9 31      LDA #$31     '1' Zweites Exponentzeichen
977E   9D 02 01   STA $0102,X  Exponent auf '01' setzen
```

```
9781   60         RTS

9782   BD 00 01   LDA $0100,X  Ziffer laden
9785   FE 00 01   INC $0100,X  Ziffer erhöhen
9788   C9 39      CMP #$39     '9' überschritten ?
978A   60         RTS

****************************** Zeichen aus Formatstring holen

978B   18         CLC
978C   C8         INY          Ende erreicht ?
978D   F0 05      BEQ $9794    Ja, Skip
978F   CC 35 01   CPY $0135    Ende erreicht ?
9792   90 04      BCC $9798    Nein, Skip
9794   A4 77      LDY $77      Rückkehr zur übergeordneten Routine ?
9796   D0 D4      BNE $976C    Ja, Ende
9798   20 D3 42   JSR $42D3    Zeichen aus Formatstring holen
979B   EE 31 01   INC $0131    Zeichenzähler erhöhen
979E   60         RTS

****************************** Zeiger vorbesetzen

979F   20 81 87   JSR $8781    FRESTR aufrufen
97A2   85 78      STA $78      Länge setzen
97A4   A2 0A      LDX #$0A     Offset laden
97A6   A9 00      LDA #$00
97A8   9D 27 01   STA $0127,X  Alle Zähler und Flags löschen
97AB   CA         DEX
97AC   10 FA      BPL $97A8
97AE   8E 26 01   STX $0126    Kommaflag löschen
97B1   86 80      STX $80
97B3   8E 25 01   STX $0125    und Dollarflag auch löschen
97B6   AA         TAX          (X) und (Y) = 0
97B7   A8         TAY
97B8   60         RTS

****************************** Zahl runden

97B9   18         CLC
97BA   A5 80      LDA $80      Kommaposition laden
97BC   6D 2D 01   ADC $012D
97BF   B0 39      BCS $97FA    Wenn Wert unpassend, Ende
97C1   38         SEC
97C2   E5 77      SBC $77
97C4   90 34      BCC $97FA    Wenn Wert unpassend, Ende
97C6   CD 24 01   CMP $0124    > Nummernende ?
97C9   F0 02      BEQ $97CD    Nein, Skip
97CB   B0 2D      BCS $97FA    Ja, Ende
```

```
97CD   CD 23 01   CMP $0123     < Nummernstart ?
97D0   90 28      BCC $97FA     Ja, Ende
97D2   AA         TAX           Wert als Offset laden
97D3   BD 00 01   LDA $0100,X   Ziffer holen
97D6   C9 35      CMP #$35      '5' ? (Aufrunden)
97D8   90 20      BCC $97FA     Nein, Ende
97DA   EC 23 01   CPX $0123     Nummernanfang erreicht ?
97DD   F0 0A      BEQ $97E9     Ja, Skip
97DF   CA         DEX           Voranstehende Ziffer anwählen
97E0   20 82 97   JSR $9782     Ziffer testen
97E3   8E 24 01   STX $0124     Zeiger auf Nummernende setzen
97E6   F0 F2      BEQ $97DA     Wenn Korrektur nötig, weitermachen
97E8   60         RTS

97E9   A9 31      LDA #$31      '1'
97EB   9D 00 01   STA $0100,X   in Nummernfeld setzen
97EE   E8         INX           Offset erhöhen
97EF   86 80      STX $80       Kommaposition setzen
97F1   C6 77      DEC $77       Länge korrigieren
97F3   10 05      BPL $97FA     Wenn korrekt, Ende
97F5   E6 77      INC $77       Länge auf 0
97F7   EE 2A 01   INC $012A     Eine Vorkommastelle mehr
97FA   60         RTS

****************************** Zahl anpassen

97FB   A4 80      LDY $80       Komma an erster Stelle ?
97FD   F0 17      BEQ $9816     Ja, Skip
97FF   AC 23 01   LDY $0123     Zeiger auf Nummernanfang setzen

9802   B9 00 01   LDA $0100,Y   Aktuelle Ziffer
9805   C9 30      CMP #$30      auf '0' testen
9807   60         RTS

9808   E6 80      INC $80       Kommaposition korrigieren
980A   20 3D 97   JSR $973D     Ziffer anpassen
980D   EE 23 01   INC $0123     Nummernanfang korrigieren
9810   CC 24 01   CPY $0124     Nummernende erreicht ?
9813   F0 E5      BEQ $97FA     Ja, Ende
9815   C8         INY           Offset erhöhen
9816   20 02 98   JSR $9802     Ziffer = '0' ?
9819   F0 ED      BEQ $9808     Ja, Zahl anpassen
981B   60         RTS
```


```
9810   CC 24 01   CPY $0124     Nummernende erreicht ?
9813   F0 E5      BEQ $97FA     Ja, Ende
9815   C8         INY           Offset erhöhen
9816   20 02 98   JSR $9802     Ziffer = '0' ?
```

```
9819   F0 ED      BEQ $9808    Ja, Zahl anpassen
981B   60         RTS

****************************** Zahl formatiert ausgeben

981C   AD 25 01   LDA $0125    Dollarzeichen ?
981F   30 02      BMI $9823    Nein, Skip
9821   E6 77      INC $77      Länge erhöhen
9823   AE 23 01   LDX $0123    Anfangszeiger setzen
9826   CA         DEX          Zähler erniedrigen
9827   AC 34 01   LDY $0134    Zeiger auf Feldbeginn setzen
982A   20 D3 42   JSR $42D3    Zeichen aus String holen
982D   C8         INY          Offset erhöhen
982E   C9 2C      CMP #$2C     ',' ?
9830   D0 14      BNE $9846    Nein, Skip
9832   2C 26 01   BIT $0126    Füllzeichenflag gesetzt ?
9835   30 09      BMI $9840    Ja, Skip
9837   8D 03 FF   STA $FF03    Read aus Bank 0 setzen
983A   AD 05 12   LDA $1205    Kommazeichen laden
983D   4C AB 98   JMP $98AB    Zeichen ausgeben

9840   AD 33 01   LDA $0133    Füllzeichen laden
9843   4C AB 98   JMP $98AB    und ausgeben

9846   C9 2E      CMP #$2E     '.' ?
9848   D0 09      BNE $9853    Nein, Skip
984A   8D 03 FF   STA $FF03    Read aus Bank 0 setzen
984D   AD 06 12   LDA $1206    Dezimalpunktzeichen laden
9850   4C AB 98   JMP $98AB    und ausgeben

9853   C9 2B      CMP #$2B     '+' ?
9855   F0 3B      BEQ $9892    Ja, Skip
9857   C9 2D      CMP #$2D     '-' ?
9859   F0 32      BEQ $988D    Ja, Skip
985B   C9 5E      CMP #$5E     '^' ?
985D   D0 69      BNE $98C8    Nein, Skip
985F   A9 45      LDA #$45     'E'
9861   20 EB 98   JSR $98EB    Zeichen auf Schirm
9864   AC 29 01   LDY $0129    Zeiger auf Exponenten setzen
9867   20 02 98   JSR $9802    Zeichen mit '0' vergleichen
986A   D0 06      BNE $9872    Keine '0', Skip
986C   C8         INY          Zeichen überlesen
986D   20 02 98   JSR $9802    Exponent ganz = 0 ?
9870   F0 07      BEQ $9879    Ja, Skip
9872   A9 2D      LDA #$2D     '-'
9874   2C 28 01   BIT $0128    Vorzeichen des Exponenten negativ ?
9877   30 02      BMI $987B    Ja, Skip
9879   A9 2B      LDA #$2B     '+'
```

```
987B   20 EB 98   JSR $98EB     Zeichen auf Schirm
987E   AE 29 01   LDX $0129     Zeiger auf Exponenten laden
9881   BD 00 01   LDA $0100,X   Zeichen aus Puffer laden
9884   20 EB 98   JSR $98EB     Zeichen auf Schirm
9887   AC 36 01   LDY $0136     Zeiger auf Feldende setzen
988A   4C A1 98   JMP $98A1     und weiter ausgeben

988D   AD 32 01   LDA $0132     Vorzeichen gesetzt ?
9890   30 AE      BMI $9840     Nein, Füllzeichen ausgeben
9892   AD 32 01   LDA $0132     Vorzeichen laden
9895   4C AB 98   JMP $98AB     und ausgeben

9898   A5 77      LDA $77       Führende Nullen ?
989A   D0 18      BNE $98B4     Nein, Skip
989C   EC 24 01   CPX $0124     Endenummer erreicht ?
989F   F0 05      BEQ $98A6     Ja, '0' ausgeben
98A1   E8         INX           Offset erhöhen
98A2   BD 00 01   LDA $0100,X   Aktuelles Zeichen laden
98A5   2C         .BYTE $2C     Nächsten Befehl überlesen
98A6   A9 30      LDA #$30      '0'
98A8   4E 26 01   LSR $0126
98AB   20 EB 98   JSR $98EB     Zeichen auf Schirm
98AE   F0 03      BEQ $98B3     Wenn Druckfeldende erreicht, Ende
98B0   4C 2A 98   JMP $982A     Zum Schleifenstart
98B3   60         RTS

98B4   C6 77      DEC $77
98B6   AD 25 01   LDA $0125     Dollarflag gesetzt ?
98B9   30 EB      BMI $98A6     Nein, Skip
98BB   38         SEC
98BC   6E 25 01   ROR $0125     Dollarflag löschen
98BF   8D 03 FF   STA $FF03     Read aus Bank 0 setzen
98C2   AD 07 12   LDA $1207     Dezimalpunktzeichen
98C5   4C A8 98   JMP $98A8     ausgeben

98C8   AD 27 01   LDA $0127     Zähler schon 0 ?
98CB   F0 CB      BEQ $9898     Ja, Wert ausgeben
98CD   CE 27 01   DEC $0127     Zäher erniedrigen
98D0   F0 03      BEQ $98D5     Jetzt 0, Skip
98D2   4C 40 98   JMP $9840     Füllzeichen ausgeben
98D5   AD 2E 01   LDA $012E     Vorzeichenflag gesetzt ?
98D8   30 F6      BMI $98D0     Nein, Skip
98DA   20 D3 42   JSR $42D3     Aktuelles Zeichen holen
98DD   C9 2C      CMP #$2C      ',' ?
98DF   D0 AC      BNE $988D     Nein, Vorzeichen ausgeben
98E1   AD 33 01   LDA $0133     Füllzeichen laden
98E4   20 EB 98   JSR $98EB     Zeichen auf Schirm
98E7   C8         INY           Offset erhöhen
```

```
98E8   4C DA 98   JMP $98DA

****************************** Zeichenausgabe für PRINT USING

98EB   20 0C 56   JSR $560C    Zeichenausgabe
98EE   CE 31 01   DEC $0131    Zeichenzähler erniedrigen
98F1   60         RTS

****************************** Zeichen bis zum ersten '#' ausgeben
                               Falls nötig Syntaxprüfung

98F2   AC 36 01   LDY $0136    Offset laden
98F5   20 8B 97   JSR $978B    Zeichen aus Formatstring holen
98F8   20 A7 99   JSR $99A7    Spezielles Formatzeichen ?
98FB   D0 14      BNE $9911    Nein, ausgeben
98FD   8C 34 01   STY $0134    (Y) retten
9900   90 1A      BCC $991C    Wenn '#', Skip
9902   AA         TAX          Zeichen retten
9903   20 8B 97   JSR $978B    Nächstes Zeichen holen
9906   B0 05      BCS $990D    Wenn Ende erreicht, Skip
9908   20 AF 99   JSR $99AF    Zeichen testen
990B   F0 0A      BEQ $9917    Wenn Formatzeichen, Skip

990D   AC 34 01   LDY $0134    (Y) wieder holen
9910   8A         TXA          Zeichen wieder holen
9911   20 0C 56   JSR $560C    Zeichenausgabe
9914   4C F5 98   JMP $98F5

9917   B0 EA      BCS $9903    Wenn kein '#', weiter suchen
9919   AC 34 01   LDY $0134    (Y) wieder holen

991C   A6 77      LDX $77      Ausgabe ?
991E   D0 7A      BNE $999A    Ja, Ende

9920   8E 31 01   STX $0131    Zeichenzähler auf 0
9923   88         DEY          Offset erniedrigen
9924   CE 31 01   DEC $0131    Zeichenzähler erniedrigen
9927   20 8B 97   JSR $978B    Zeichen aus Formatstring holen
992A   B0 74      BCS $99A0    Wenn Ende erreicht, Skip
992C   C9 2C      CMP #$2C     ',' ?
992E   F0 F7      BEQ $9927    Ja, weitertesten
9930   20 7E 99   JSR $997E    Test auf Vorzeichen
9933   90 EF      BCC $9924    Wenn Vorzeichen, weitertesten
9935   C9 2E      CMP #$2E     '.' ?
9937   D0 08      BNE $9941    Nein, Skip
9939   E8         INX          Kommazähler erhöhen
993A   E0 02      CPX #$02     Zwei Kommas gelesen ?
993C   90 E9      BCC $9927    Nein, weitertesten
```

```
993E   4C 6C 79   JMP $796C     'SYNTAX'

9941   20 B3 99   JSR $99B3     Stringformatzeichen ?
9944   D0 0B      BNE $9951     Nein, Skip
9946   90 03      BCC $994B     Wenn '#', Skip
9948   8D 2B 01   STA $012B
994B   FE 2C 01   INC $012C,X   Stellenzähler erhöhen
994E   4C 27 99   JMP $9927     und weitertesten
9951   C9 24      CMP #$24      '$' ?
9953   D0 0F      BNE $9964     Nein, Skip
9955   2C 25 01   BIT $0125     Dollarflag schon gesetzt ?
9958   10 F1      BPL $994B     Ja, Zeichen als Druckzeichen angeben
995A   18         CLC
995B   6E 25 01   ROR $0125     Dollarflag setzen (Bit 7 = 0)
995E   CE 2C 01   DEC $012C     Vorkommastellenzähler erniedrigen
9961   4C 4B 99   JMP $994B     und weitermachen
9964   C9 5E      CMP #$5E      '^' ?
9966   D0 16      BNE $997E     Nein, Skip
9968   A2 02      LDX #$02      Die nächsten drei Zeichen auch '^' ?
996A   20 8B 97   JSR $978B     Zeichen aus Formatstring holen
996D   B0 CF      BCS $993E     Kein '^', Fehler
996F   C9 5E      CMP #$5E      '^' ?
9971   D0 CB      BNE $993E     Nein, Fehler
9973   CA         DEX           Alle Zeichen getestet ?
9974   10 F4      BPL $996A     Nein, nochmal
9976   EE 2F 01   INC $012F     Exponentflag setzen
9979   20 8B 97   JSR $978B     Zeichen aus Formatstring holen
997C   B0 22      BCS $99A0     Ende, Skip

997E   C9 2B      CMP #$2B      '+' ?
9980   D0 19      BNE $999B     Nein, Skip
9982   AD 32 01   LDA $0132     Vorzeichen bereits gesetzt ?
9985   10 05      BPL $998C     Ja, Skip
9987   A9 2B      LDA #$2B      '+'
9989   8D 32 01   STA $0132
998C   AD 2E 01   LDA $012E     Vorzeichenflag schon gesetzt ?
998F   D0 AD      BNE $993E     Ja, Fehler
9991   6E 2E 01   ROR $012E     Bit 7 im Vorzeichenflag setzen
9994   8C 36 01   STY $0136     Zeiger auf Feldende setzen
9997   EE 31 01   INC $0131     und Zeichenzählererhöhen
999A   60         RTS

999B   C9 2D      CMP #$2D      '-' ?
999D   F0 ED      BEQ $998C     Ja, Vorzeichen verarbeiten
999F   38         SEC
99A0   8C 36 01   STY $0136     Zeiger auf Feldende setzen
99A3   CE 36 01   DEC $0136     und korrigieren
```

```
99A6   60          RTS

****************************** Test auf Vorzeichen und Formatzeichen

99A7   C9 2B       CMP #$2B      '+' ?
99A9   F0 15       BEQ $99C0     Ja, Skip
99AB   C9 2D       CMP #$2D      '-' ?
99AD   F0 11       BEQ $99C0     Ja, Skip

99AF   C9 2E       CMP #$2E      '.' ?
99B1   F0 0D       BEQ $99C0     Ja, Skip

99B3   C9 3D       CMP #$3D      '=' ?
99B5   F0 09       BEQ $99C0     Ja, Skip
99B7   C9 3E       CMP #$3E      '>' ?
99B9   F0 05       BEQ $99C0     Ja, Skip
99BB   C9 23       CMP #$23      '#' ?
99BD   D0 01       BNE $99C0     Nein, Skip
99BF   18          CLC           Für '#' BEQ + CLC
99C0   60          RTS

****************************** BASIC-Funktion INSTR

99C1   A5 66       LDA $66       Quellstringadresse setzen
99C3   8D D6 03    STA $03D6
99C6   A5 67       LDA $67
99C8   8D D7 03    STA $03D7
99CB   20 EF 77    JSR $77EF     FRMEVL Ausdruck auswerten
99CE   20 DD 77    JSR $77DD     Auf String prüfen
99D1   A5 66       LDA $66       Suchstringadresse setzen
99D3   8D D8 03    STA $03D8
99D6   A5 67       LDA $67
99D8   8D D9 03    STA $03D9
99DB   A2 01       LDX #$01      Suchstart auf 1 setzen
99DD   86 67       STX $67
99DF   20 86 03    JSR $0386     CHRGOT
99E2   C9 29       CMP #$29      = ')' ?
99E4   F0 03       BEQ $99E9     Ja, Skip
99E6   20 09 88    JSR $8809     Byte-Wert nach Komma holen
99E9   20 56 79    JSR $7956     Test auf ')'
99EC   A6 67       LDX $67       Suchstart = 0 ?
99EE   D0 03       BNE $99F3     Nein, Skip
99F0   4C 28 7D    JMP $7D28     'ILLEGAL QUANTITY'
99F3   CA          DEX
99F4   86 63       STX $63       Suchposition setzen
99F6   A2 03       LDX #$03      Stringadressen umkopieren
99F8   BD D6 03    LDA $03D6,X
99FB   95 59       STA $59,X
```

```
99FD   CA         DEX
99FE   10 F8      BPL $99F8
9A00   A0 02      LDY #$02     Ein Deskriptor hat 3 Bytes
9A02   A9 59      LDA #$59     Aus Adresse ($59)
9A04   20 AB 03   JSR $03AB    Deskriptor für Quellstring lesen
9A07   99 5D 00   STA $005D,Y
9A0A   A9 5B      LDA #$5B     Aus Adresse ($5B)
9A0C   20 AB 03   JSR $03AB    Deskriptor für Suchstring lesen
9A0F   99 60 00   STA $0060,Y
9A12   88         DEY          Alle Bytes kopiert ?
9A13   10 ED      BPL $9A02    Nein, nochmal
9A15   A5 60      LDA $60      Suchstringlänge = 0 ?
9A17   F0 3B      BEQ $9A54    Ja, nicht gefunden, Ende
9A19   A9 00      LDA #$00     Suchzeiger setzen
9A1B   85 64      STA $64
9A1D   18         CLC
9A1E   A5 60      LDA $60      Suchstringlänge
9A20   65 63      ADC $63      + Suchposition > 255
9A22   B0 30      BCS $9A54    Ja, nicht gefunden, Ende
9A24   C5 5D      CMP $5D      > Quellstringlänge ?
9A26   90 02      BCC $9A2A    Nein, Skip
9A28   D0 2A      BNE $9A54    Ja, nicht gefunden, Ende
9A2A   A4 64      LDY $64      Suchzeiger
9A2C   C4 60      CPY $60      = Suchstringlänge ?
9A2E   F0 1F      BEQ $9A4F    Ja, gefunden, Ende
9A30   98         TYA
9A31   18         CLC
9A32   65 63      ADC $63      (Y) = Suchzeiger + Suchposition
9A34   A8         TAY
9A35   A9 5E      LDA #$5E     Ergibt LDA ($5E),Y aus Bank 1
9A37   20 AB 03   JSR $03AB    Zeichen aus Quellstring holen
9A3A   85 79      STA $79      und setzen
9A3C   A4 64      LDY $64      (Y) = Suchzeiger
9A3E   A9 61      LDA #$61     Ergibt LDA ($61),Y aus Bank 1
9A40   20 AB 03   JSR $03AB    Zeichen aus Suchstring holen
9A43   C5 79      CMP $79      = Zeichen aus Quellstring ?
9A45   F0 04      BEQ $9A4B    Ja, Skip
9A47   E6 63      INC $63      Suchposition erhöhen
9A49   D0 CE      BNE $9A19    weitertesten
9A4B   E6 64      INC $64      Suchzeiger erhöhen
9A4D   D0 DB      BNE $9A2A    weiter testen
9A4F   E6 63      INC $63      Suchposition anpassen
9A51   A5 63      LDA $63      und als Ergebnis laden
9A53   2C         .BYTE $2C    Nächsten Befehl überlesen
9A54   A9 00      LDA #$00     Wert für String nicht gefunden
9A56   8D 03 FF   STA $FF03    Write in Bank 1 setzen
9A59   48         PHA          Ergebnis retten
9A5A   AD D8 03   LDA $03D8    Adresse des Quellstrings
```

```
96C9   F0 0B      BEQ $96D6   Nein, Überlauf
96CB   88         DEY         Restplatz vorhanden ?
96CC   D0 13      BNE $96E1   Ja, Skip
96CE   AD 2D 01   LDA $012D   Möglichkeiten ausreichend ?
96D1   0D 2A 01   ORA $012A
96D4   D0 AC      BNE $9682   Ja, Zahl ausgeben
96D6   A9 2A      LDA #$2A    '*'
96D8   20 EB 98   JSR $98EB   ausgeben
96DB   D0 FB      BNE $96D8   Wenn Ende noch nicht erreicht, mehr '*'
96DD   60         RTS

96DE   A8         TAY         Restplatz vorhanden ?
96DF   F0 A1      BEQ $9682   Nein, Zahl ausgeben
96E1   AD 2A 01   LDA $012A   Vorkommastellen vorhanden ?
96E4   D0 9C      BNE $9682   Ja, Zahl ausgeben
96E6   CE 27 01   DEC $0127   Restplatz korrigieren
96E9   E6 77      INC $77     und Zähler erhöhen
96EB   4C 82 96   JMP $9682   Zahl ausgeben

****************************** Stellenzahl anpassen

96EE   38         SEC
96EF   AD 2C 01   LDA $012C   Feldpositionen für Vorkommastellen
96F2   ED 2A 01   SBC $012A   = Vorkommastellenzahl ?
96F5   F0 39      BEQ $9730   Ja, Ende
96F7   A0 80      LDY $80     Zeiger auf Dezimalpunkt laden
96F9   90 16      BCC $9711   Wenn zuwenig Stellen, Skip
96FB   85 78      STA $78     Anzahl der überflüssigen Positionen
setzen
96FD   CC 24 01   CPY $0124   Nummerende erreicht ?
9700   F0 02      BEQ $9704   Ja, Skip
9702   B0 01      BCS $9705   Wenn größer, Skip
9704   C8         INY         Kommaposition korrigieren
9705   EE 2A 01   INC $012A   Vorkommastellen erhöhen
9708   20 3D 97   JSR $973D   Exponent berechnen
970B   C6 78      DEC $78     Alle Stellen aufgearbeitet ?
970D   D0 EE      BNE $96FD   Nein, weitermachen
970F   F0 1D      BEQ $972E   Unbedingter Sprung

9711   49 FF      EOR #$FF    Stellenmangel positiv machen
9713   69 01      ADC #$01
9715   85 78      STA $78     und setzen
9717   CC 23 01   CPY $0123   Anfang erreicht ?
971A   F0 07      BEQ $9723   Ja, Skip
971C   88         DEY         Kommastelle korrigieren
971D   CE 2A 01   DEC $012A   Vorkommastellen erniedrigen
9720   4C 25 97   JMP $9725
9723   E6 77      INC $77     Zähler erhohen
```

```
9AAC   30 0C      BMI $9ABA    Nein, Skip
9AAE   7D 3E 9F   ADC $9F3E,X  Feinwinkelwert aufaddieren
9AB1   48         PHA          und Low-Byte retten
9AB2   98         TYA          High-Byte laden
9AB3   7D 3D 9F   ADC $9F3D,X  Feinwinkelwert aufaddieren
9AB6   A8         TAY          und High-byte wieder in (Y)
9AB7   68         PLA          Low-Byte wieder holen
9AB8   90 EF      BCC $9AA9    Unbedingter Sprung
9ABA   48         PHA          Low-Byte retten
9ABB   A2 00      LDX #$00     Offset für Sinus
9ABD   AD 49 11   LDA $1149    Ist ein Sinus ausgerechnet worden ?
9AC0   4A         LSR          Bit 0 ist wichtig !
9AC1   B0 02      BCS $9AC5    Ja, Skip
9AC3   A2 02      LDX #$02     Offset für Cosinus
9AC5   68         PLA          Low-Byte holen
9AC6   9D 4A 11   STA $114A,X  Winkelfunktionswert eintragen
9AC9   98         TYA          High-Byte holen
9ACA   9D 4B 11   STA $114B,X  und Wert eintragen
9ACD   60         RTS

****************************** SIN/COS mit Koordinate multiplizieren

9ACE   A0 19      LDY #$19     Offset für Sinuswert laden
9AD0   90 02      BCC $9AD4    Wenn Sinus multiplizieren, Skip
9AD2   A0 1B      LDY #$1B     Offset für Cosinuswert
9AD4   AD 49 11   LDA $1149    Quadrant
9AD7   69 02      ADC #$02     + 2
9AD9   4A         LSR
9ADA   4A         LSR          Carry = Flag Quadrant 1 oder 4
9ADB   08         PHP
9ADC    20 8F 9D    JSR $9D8F      Koordinate mit Offset (Y) in (A)/(Y)
laden
9ADF   C0 FF      CPY #$FF     Wert unpassend ?
9AE1   90 07      BCC $9AEA    Nein, Skip
9AE3   8A         TXA          Koordinatennummer in (Y) bringen
9AE4   A8         TAY
9AE5   20 8F 9D   JSR $9D8F    Koordinate laden
9AE8   B0 03      BCS $9AED    Unbedingter Sprung
9AEA   20 AE 9D   JSR $9DAE    Koordinate mit Winkelwert multiplizieren
9AED   28         PLP          Quadrant 1 oder 4 ?
9AEE   B0 1B      BCS $9B0B    Ja, Ende
9AF0   4C 9E 9D   JMP $9D9E    Ergebnis negieren

****************************** Abstandswerte berechnen

9AF3   8D 4E 11   STA $114E    SIN/COS-Flag setzen
9AF6   A2 23      LDX #$23     Offset auf Abstandsspeicher
9AF8   0E 4E 11   ASL $114E    SIN/COS-Bit setzen
```

```
9AFB    20 CE 9A    JSR $9ACE      SIN/COS-Wert multiplizieren
9AFE    9D 31 11    STA $1131,X    Abstandswerte von $1154-$115B setzen
9B01    98          TYA            High-Byte laden
9B02    9D 32 11    STA $1132,X    und setzen
9B05    E8          INX            Offset erhöhen
9B06    E8          INX            Offset erhöhen
9B07    E0 2B       CPX #$2B       Beide Abstände ausgerechnet ?
9B09    90 ED       BCC $9AF8      Nein, nochmal
9B0B    60          RTS

****************************** BASIC-Befehl RDOT

9B0C    20 F7 87    JSR $87F7      Byte-Wert holen
9B0F    E0 02       CPX #$02       = 2 ?
9B11    90 10       BCC $9B23      Wenn < 2, Skip
9B13    F0 03       BEQ $9B18      Wenn = 2, Skip
9B15    4C 28 7D    JMP $7D28      'ILLEGAL QUANTITY'

9B18    20 49 9C    JSR $9C49      Farbwert an aktuellen Koordinaten lesen
9B1B    A8          TAY            Wert in (Y)
9B1C    90 02       BCC $9B20      Wenn kein Fehler, Skip
9B1E    A0 00       LDY #$00       Sonst Wert für Hintergrundfarbe
9B20    4C D4 84    JMP $84D4      in FAC#1

9B23    8A          TXA            Offset in (A)
9B24    0A          ASL            = Wert * 2
9B25    AA          TAX            Als Offset laden
9B26    BD 31 11    LDA $1131,X    Und aktuelle X- oder Y-Koordinate
auslesen
9B29    A8          TAY            Low-Byte in (Y) retten
9B2A    BD 32 11    LDA $1132,X    High-Byte laden
9B2D    4C 3C 79    JMP $793C      und in FAC#1

****************************** Linie von aktueller Koordinate zu
                               Zielkoordinate ziehen

9B30    A2 02       LDX #$02       Offset auf Y-Vorzeichen
9B32    A0 06       LDY #$06       Offset auf Y-Zielkoordinate
9B34    A9 00       LDA #$00
9B36    9D 3D 11    STA $113D,X    Vorzeichen auf $0000
9B39    9D 3E 11    STA $113E,X    setzen
9B3C    20 99 9D    JSR $9D99      (A)/(Y) = ABS(Start - Ziel)
9B3F    10 08       BPL $9B49      Wenn Differnez positiv, Skip
9B41    DE 3D 11    DEC $113D,X    Vorzeichen auf $FFFF
9B44    DE 3E 11    DEC $113E,X    setzen
9B47    D0 0B       BNE $9B54      Unbedingter Skip
9B49    C9 00       CMP #$00       Ergebnis = 0 ?
9B4B    D0 04       BNE $9B51      Nein, Skip
```

```
97CD   CD 23 01   CMP $0123     < Nummernstart ?
97D0   90 28      BCC $97FA     Ja, Ende
97D2   AA         TAX           Wert als Offset laden
97D3   BD 00 01   LDA $0100,X   Ziffer holen
97D6   C9 35      CMP #$35      '5' ? (Aufrunden)
97D8   90 20      BCC $97FA     Nein, Ende
97DA   EC 23 01   CPX $0123     Nummernanfang erreicht ?
97DD   F0 0A      BEQ $97E9     Ja, Skip
97DF   CA         DEX           Voranstehende Ziffer anwählen
97E0   20 82 97   JSR $9782     Ziffer testen
97E3   8E 24 01   STX $0124     Zeiger auf Nummernende setzen
97E6   F0 F2      BEQ $97DA     Wenn Korrektur nötig, weitermachen
97E8   60         RTS

97E9   A9 31      LDA #$31      '1'
97EB   9D 00 01   STA $0100,X   in Nummernfeld setzen
97EE   E8         INX           Offset erhöhen
97EF   86 80      STX $80       Kommaposition setzen
97F1   C6 77      DEC $77       Länge korrigieren
97F3   10 05      BPL $97FA     Wenn korrekt, Ende
97F5   E6 77      INC $77       Länge auf 0
97F7   EE 2A 01   INC $012A     Eine Vorkommastelle mehr
97FA   60         RTS

******************************* Zahl anpassen

97FB   A4 80      LDY $80       Komma an erster Stelle ?
97FD   F0 17      BEQ $9816     Ja, Skip
97FF   AC 23 01   LDY $0123     Zeiger auf Nummernanfang setzen

9802   B9 00 01   LDA $0100,Y   Aktuelle Ziffer
9805   C9 30      CMP #$30      auf '0' testen
9807   60         RTS

9808   E6 80      INC $80       Kommaposition korrigieren
980A   20 3D 97   JSR $973D     Ziffer anpassen
980D   EE 23 01   INC $0123     Nummernanfang korrigieren
9810   CC 24 01   CPY $0124     Nummernende erreicht ?
9813   F0 E5      BEQ $97FA     Ja, Ende
9815   C8         INY           Offset erhöhen
9816   20 02 98   JSR $9802     Ziffer = '0' ?
9819   F0 ED      BEQ $9808     Ja, Zahl anpassen
981B   60         RTS
```


```
9810   CC 24 01   CPY $0124     Nummernende erreicht ?
9813   F0 E5      BEQ $97FA     Ja, Ende
9815   C8         INY           Offset erhöhen
9816   20 02 98   JSR $9802     Ziffer = '0' ?
```

```
9819   F0 ED      BEQ $9808    Ja, Zahl anpassen
981B   60         RTS

****************************** Zahl formatiert ausgeben

981C   AD 25 01   LDA $0125    Dollarzeichen ?
981F   30 02      BMI $9823    Nein, Skip
9821   E6 77      INC $77      Länge erhöhen
9823   AE 23 01   LDX $0123    Anfangszeiger setzen
9826   CA         DEX          Zähler erniedrigen
9827   AC 34 01   LDY $0134    Zeiger auf Feldbeginn setzen
982A   20 D3 42   JSR $42D3    Zeichen aus String holen
982D   C8         INY          Offset erhöhen
982E   C9 2C      CMP #$2C     ',' ?
9830   D0 14      BNE $9846    Nein, Skip
9832   2C 26 01   BIT $0126    Füllzeichenflag gesetzt ?
9835   30 09      BMI $9840    Ja, Skip
9837   8D 03 FF   STA $FF03    Read aus Bank 0 setzen
983A   AD 05 12   LDA $1205    Kommazeichen laden
983D   4C AB 98   JMP $98AB    Zeichen ausgeben

9840   AD 33 01   LDA $0133    Füllzeichen laden
9843   4C AB 98   JMP $98AB    und ausgeben

9846   C9 2E      CMP #$2E     '.' ?
9848   D0 09      BNE $9853    Nein, Skip
984A   8D 03 FF   STA $FF03    Read aus Bank 0 setzen
984D   AD 06 12   LDA $1206    Dezimalpunktzeichen laden
9850   4C AB 98   JMP $98AB    und ausgeben

9853   C9 2B      CMP #$2B     '+' ?
9855   F0 3B      BEQ $9892    Ja, Skip
9857   C9 2D      CMP #$2D     '-' ?
9859   F0 32      BEQ $988D    Ja, Skip
985B   C9 5E      CMP #$5E     '^' ?
985D   D0 69      BNE $98C8    Nein, Skip
985F   A9 45      LDA #$45     'E'
9861   20 EB 98   JSR $98EB    Zeichen auf Schirm
9864   AC 29 01   LDY $0129    Zeiger auf Exponenten setzen
9867   20 02 98   JSR $9802    Zeichen mit '0' vergleichen
986A   D0 06      BNE $9872    Keine '0', Skip
986C   C8         INY          Zeichen überlesen
986D   20 02 98   JSR $9802    Exponent ganz = 0 ?
9870   F0 07      BEQ $9879    Ja, Skip
9872   A9 2D      LDA #$2D     '-'
9874   2C 28 01   BIT $0128    Vorzeichen des Exponenten negativ ?
9877   30 02      BMI $987B    Ja, Skip
9879   A9 2B      LDA #$2B     '+'
```

```
9C08  EE 32 11   INC $1132     Übertrag berücksichtigen
9C0B  20 19 9C   JSR $9C19     Punkt setzen
9C0E  AE 31 11   LDX $1131     X-Koordinate erniedrigen
9C11  D0 03      BNE $9C16     Kein Übertrag, Skip
9C13  CE 32 11   DEC $1132     Übertrag berücksichtigen
9C16  CE 31 11   DEC $1131     Low-Byte erniedrigen

****************************** Punkt nach Farbquelle setzen

9C19  20 24 9D   JSR $9D24     Graphikkoordinaten auf Schirm umrechnen
9C1C  B0 24      BCS $9C42     Wenn Fehler, Skip
9C1E  20 70 9C   JSR $9C70     Farbwert in Video-RAM setzen
9C21  20 E8 9C   JSR $9CE8     Schirmadresse berechnen
9C24  8D 6D 11   STA $116D     Bitmuster speichern
9C27  B1 8C      LDA ($8C),Y   Zeichen aus Schirm holen
9C29  0D 6D 11   ORA $116D     Bit setzen
9C2C  24 D8      BIT $D8       Multicolormodus ?
9C2E  10 13      BPL $9C43     Nein, Skip
9C30  48         PHA           Zeichenwert retten
9C31  A6 83      LDX $83       Bitmuster nach Farbquelle korrigieren
9C33  AD 6D 11   LDA $116D
9C36  3D 25 9F   AND $9F25,X
9C39  8D 6D 11   STA $116D
9C3C  68         PLA           Zeichen wieder holen
9C3D  4D 6D 11   EOR $116D     mit Bitmuster verknüpfen
9C40  91 8C      STA ($8C),Y   Bitmuster in Schirm setzen
9C42  60         RTS

9C43  A6 83      LDX $83       Farbquelle gesetzt ?
9C45  D0 F9      BNE $9C40     Ja, Bitmuster in Schirm
9C47  F0 F4      BEQ $9C3D     Nein, Bit löschen und in Schirm

****************************** Koordinaten testen (für PAINT)

9C49  20 E3 9C   JSR $9CE3     Schirmadresse berechnen
9C4C  B0 21      BCS $9C6F     Wenn Fehler, Skip
9C4E  8D 6D 11   STA $116D     Bitmuster setzen
9C51  B1 8C      LDA ($8C),Y   Bitmuster aus Schirm holen
9C53  2D 6D 11   AND $116D     Gesuchtes Bitmuster isolieren
9C56  2A         ROL           und an Anfang von (A) bringen
9C57  CA         DEX           Zähler erniedrigen
9C58  10 FC      BPL $9C56     Bitmuster weiter schieben, wenn nötig
9C5A  2A         ROL           Bit in C-Flag schieben
9C5B  24 8B      BIT $8B       Vergleich auf Hintergrundfarbe ?
9C5D  30 06      BMI $9C65     Ja, Skip
9C5F  29 03      AND #$03      Bitmuster normalisieren
9C61  C5 83      CMP $83       = gewählte Farbe ?
9C63  18         CLC
```

```
9C64   60         RTS

9C65   18         CLC
9C66   29 03      AND #$03      Bitmuster normalisieren
9C68   F0 03      BEQ $9C6D     Wenn Hintergrundfarbe, Skip
9C6A   A2 00      LDX #$00      BEQ
9C6C   60         RTS

9C6D   A2 FF      LDX #$FF      BNE
9C6F   60         RTS

***************************** Aktuellen Farbwert in Zeile (X)
                              und Spalte (Y) bringen

9C70   BD 33 C0   LDA $C033,X   Videoadresse im HIRES-Modus setzen
9C73   85 8C      STA $8C       Low-Byte setzen
9C75   BD CA 9C   LDA $9CCA,X   High-Byte für HIRES-Farbschirm laden
9C78   85 8D      STA $8D       High-Byte setzen
9C7A   A5 83      LDA $83       Farbquelle = Hintergrund ?
9C7C   D0 08      BNE $9C86     Nein, Skip
9C7E   AD E2 03   LDA $03E2     Hintergrundnibble laden
9C81   24 D8      BIT $D8       Mehrfarbmodus ?
9C83   10 08      BPL $9C8D     Nein, Skip
9C85   60         RTS
9C86   C9 02      CMP #$02      Farbquelle = Zusatzfarbe 1 ?
9C88   D0 10      BNE $9C9A     Nein, Skip
9C8A   AD E3 03   LDA $03E3     Multicolornibble 1 laden
9C8D   29 0F      AND #$0F      und Farbnibble isolieren
9C8F   85 77      STA $77       Farbnibble setzen
9C91   B1 8C      LDA ($8C),Y   Farbwert aus Bildschirm laden
9C93   29 F0      AND #$F0      oberes Nibble isolieren
9C95   05 77      ORA $77       und mit Farbnibble verknüpfen
9C97   91 8C      STA ($8C),Y   Farbwert wieder setzen
9C99   60         RTS
9C9A   B0 10      BCS $9CAC     Wenn Farbquelle > 2, Skip
9C9C   AD E2 03   LDA $03E2     Multicolornibble 2 laden
9C9F   29 F0      AND #$F0      Oberes Nibble isolieren
9CA1   85 77      STA $77       und setzen
9CA3   B1 8C      LDA ($8C),Y   Farbwert aus Bildschirm laden
9CA5   29 0F      AND #$0F      Unteres Nibble isolieren
9CA7   05 77      ORA $77       und mit Farbnibble verknüpfen
9CA9   91 8C      STA ($8C),Y   Farbwert wieder setzen
9CAB   60         RTS
9CAC   A5 8D      LDA $8D       Adreßzeiger auf Farb-RAM setzen
9CAE   29 03      AND #$03      Wichtige Bits retten
9CB0   09 D8      ORA #$D8      High-Byte auf $D800 setzen
9CB2   85 8D      STA $8D       und speichern
9CB4   A9 00      LDA #$00      Alle ROMs und I/O-Bereich anschalten
```

```
9CB6   8D 00 FF   STA $FF00
9CB9   78         SEI          Schirmverwaltung ausschalten
9CBA   A5 01      LDA $01      Farb-RAM-Auswahl retten
9CBC   48         PHA
9CBD   29 FE      AND #$FE     Farb-RAM 2 für Graphik anwählen
9CBF   85 01      STA $01
9CC1   A5 85      LDA $85      Multicolorfarbe 2
9CC3   91 8C      STA ($8C),Y  in Farb-RAM setzen
9CC5   68         PLA          Auswahl wieder holen
9CC6   85 01      STA $01      Farb-RAM-Auswahl wieder setzen
9CC8   58         CLI          Schirmverwaltung wieder ein
9CC9   60         RTS

****************************** Video-RAM High-Bytes im HIRES-Modus

9CCA   1C 1C 1C 1C 1C 1C 1C 1D
9CD2   1D 1D 1D 1D 1D 1E 1E 1E
9CDA   1E 1E 1E 1E 1F 1F 1F 1F

****************************** Graphikschirmadreßwerte berechnen

9CE3   20 24 9D   JSR $9D24    Koordinate umrechnen
9CE6   B0 33      BCS $9D1B    Wenn unpassende Koordinate, Ende
9CE8   98         TYA
9CE9   18         CLC
9CEA   7D 33 C0   ADC $C033,X  ($8C) = 8 * Schirmposition
9CED   85 8C      STA $8C      entspricht Graphikschirmadresse
9CEF   BD 4C C0   LDA $C04C,X  High-Byte laden
9CF2   69 00      ADC #$00     Übertrag berücksichtigen
9CF4   06 8C      ASL $8C      Low-Byte * 2
9CF6   2A         ROL          High-Byte * 2
9CF7   06 8C      ASL $8C      Low-Byte * 4
9CF9   2A         ROL          High-Byte * 4
9CFA   06 8C      ASL $8C      Low-Byte * 8
9CFC   2A         ROL          High-Byte * 8
9CFD   85 8D      STA $8D      High-Byte setzen
9CFF   AD 33 11   LDA $1133    (Y) = Y-Wert AND 7
9D02   29 07      AND #$07
9D04   A8         TAY          Als Offset auf Byte an Position laden
9D05   AD 31 11   LDA $1131    X-Wert laden
9D08   24 D8      BIT $D8      Multicolormodus ?
9D0A   08         PHP          Status retten
9D0B   10 01      BPL $9D0E    Nein, Skip
9D0D   0A         ASL          X-Wert verdoppeln
9D0E   29 07      AND #$07     (X) = X-Wert AND 7
9D10   AA         TAX          Als Offset auf Bit laden
9D11   BD 1C 9D   LDA $9D1C,X  Bitposition aus Tabelle holen
9D14   28         PLP          Multicolormodus ?
```

```
9D15   10 04      BPL $9D1B    Nein, Skip
9D17   E8         INX          Offset erhöhen
9D18   1D 1C 9D   ORA $9D1C,X  2. Bit dazusetzen
9D1B   60         RTS

****************************** Zweierpotenzen abwärts

9D1C   80 40 20 10 08 04 02 01

****************************** Aktuelle Koordinate auf Schirmpositionen
                               umrechnen

9D24   AD 32 11   LDA $1132    X-High > 1 ?
9D27   4A         LSR
9D28   D0 1E      BNE $9D48    Ja, SEC Ende
9D2A   AD 31 11   LDA $1131
9D2D   6A         ROR          X-Wert / 4
9D2E   4A         LSR
9D2F   24 D8      BIT $D8      Mehrfarbmodus ?
9D31   30 01      BMI $9D34    Ja, Skip
9D33   4A         LSR          X-Wert / 8
9D34   A8         TAY
9D35   C0 28      CPY #$28     Schirmwert > 39 ?
9D37   B0 0F      BCS $9D48    Ja, SEC Ende
9D39   AD 34 11   LDA $1134    Y-High gesetzt ?
9D3C   D0 0A      BNE $9D48    Ja, SEC Ende
9D3E   AD 33 11   LDA $1133    Y-Wert / 8
9D41   4A         LSR
9D42   4A         LSR
9D43   4A         LSR
9D44   AA         TAX
9D45   C9 19      CMP #$19     Wenn Y-Wert > 24, SEC
9D47   60         RTS
9D48   38         SEC          SEC = Koordinaten unpassend
9D49   60         RTS

****************************** X und Y skalieren

9D4A   AD 6A 11   LDA $116A    Skalierung ?
9D4D   F0 17      BEQ $9D66    Nein, Ende
9D4F   A5 87      LDA $87      Skalierungsfaktor X laden
9D51   A4 88      LDY $88
9D53   20 5A 9D   JSR $9D5A    X-Koordinate skalieren
9D56   A5 89      LDA $89      Skalierungsfaktor Y laden
9D58   A4 8A      LDY $8A
```

```
****************************** Eine Koordinate skalieren

9D5A   20 AE 9D   JSR $9DAE     Wert mit Koordinate multiplizieren
9D5D   9D 31 11   STA $1131,X   Neuen Koordinatenwert setzen
9D60   98         TYA
9D61   E8         INX           Offset erhöhen
9D62   9D 31 11   STA $1131,X
9D65   E8         INX           Offset erhöhen
9D66   60         RTS

****************************** (A)/(Y) = (A)/(Y) +/- Koordinate 2

9D67   90 07      BCC $9D70     Addition, Skip
9D69   B0 14      BCS $9D7F     Subtraktion, Skip

****************************** (A)/(Y) = Koordinate 1 +/- Koordinate 2

9D6B   B0 0F      BCS $9D7C     Subtraktion, Skip

****************************** (A)/(Y) = Koordinate 1 + Koordinate 2

9D6D   20 8F 9D   JSR $9D8F     Aktuelle Koordinate laden
9D70   18         CLC
9D71   7D 31 11   ADC $1131,X   Low-Byte aufaddieren
9D74   48         PHA           und retten
9D75   98         TYA           High-Byte laden
9D76   7D 32 11   ADC $1132,X   Koordinate aufaddieren
9D79   A8         TAY           und High-Byte in (Y)
9D7A   68         PLA           Low-Byte wieder in (A)
9D7B   60         RTS

****************************** (A)/(Y) = Koordinate 1 - Koordinate 2

9D7C   20 8F 9D   JSR $9D8F     Aktuelle Koordinate laden
9D7F   38         SEC
9D80   FD 31 11   SBC $1131,X   Low-Byte abziehen
9D83   85 59      STA $59       und retten
9D85   98         TYA           High-Byte laden
9D86   FD 32 11   SBC $1132,X   und abziehen
9D89   A8         TAY           High-Byte wieder in (Y)
9D8A   08         PHP           Status retten
9D8B   A5 59      LDA $59       Low-Byte wieder laden
9D8D   28         PLP           und Status wieder holen
9D8E   60         RTS

****************************** Aktuelle Koordinate 1 in (A)/(Y)

9D8F   B9 31 11   LDA $1131,Y   Low-Byte laden
```

```
9D92   48         PHA          und retten
9D93   B9 32 11   LDA $1132,Y  High-Byte laden
9D96   A8         TAY          und in (Y)
9D97   68         PLA          Low-Byte wieder holen
9D98   60         RTS

****************************** (A)/(Y) = ABS(Koordinate 1 - Koordinate
2)

9D99   20 7C 9D   JSR $9D7C    Koordinaten voneinander abziehen

****************************** (A)/(Y) = ABS((A)/(Y))

9D9C   10 0F      BPL $9DAD    Wenn positiv, Skip
9D9E   08         PHP          Status retten
9D9F   18         CLC
9DA0   49 FF      EOR #$FF     Zweierkomplement des Low-Byte
9DA2   69 01      ADC #$01     errechnen
9DA4   48         PHA          und retten
9DA5   98         TYA          High-Byte holen
9DA6   49 FF      EOR #$FF     Zweierkomplement des High-Byte
9DA8   69 00      ADC #$00     errechnen
9DAA   A8         TAY          ergibt positiven
9DAB   68         PLA          Wert in (A)/(Y)
9DAC   28         PLP          Status wieder laden
9DAD   60         RTS

****************************** Koordinate mit (A)/(Y) multiplizieren

9DAE   84 8E      STY $8E      Skalierungsfaktor setzen
9DB0   85 8F      STA $8F
9DB2   BD 31 11   LDA $1131,X  Koordinate laden
9DB5   BC 32 11   LDY $1132,X
9DB8   08         PHP          Vorzeichen retten
9DB9   20 9C 9D   JSR $9D9C    Absolutwert bilden
9DBC   9D 31 11   STA $1131,X  Koordinate wieder speichern
9DBF   98         TYA          High-Byte in (A)
9DC0   9D 32 11   STA $1132,X  und High-Byte setzen
9DC3   A9 00      LDA #$00     Übertrag löschen
9DC5   8D 77 11   STA $1177
9DC8   A0 10      LDY #$10     Koordinate mit Faktor multiplizieren
9DCA   46 8E      LSR $8E
9DCC   66 8F      ROR $8F
9DCE   90 0F      BCC $9DDF
9DD0   18         CLC
9DD1   7D 31 11   ADC $1131,X
9DD4   48         PHA
9DD5   AD 77 11   LDA $1177
```

```
9DD8   7D 32 11   ADC $1132,X  Übertrag berücksichtigen
9DDB   8D 77 11   STA $1177    Übertrag setzen
9DDE   68         PLA
9DDF   4E 77 11   LSR $1177
9DE2   6A         ROR
9DE3   88         DEY          16 Bit multipliziert ?
9DE4   D0 E4      BNE $9DCA    Nein, weitermachen
9DE6   69 00      ADC #$00     Übertrag setzen
9DE8   AC 77 11   LDY $1177    High-Byte laden
9DEB   90 01      BCC $9DEE    Kein Übertrag, Skip
9DED   C8         INY          Übertrag berücksichtigen
9DEE   28         PLP          Altes Vorzeichen holen
9DEF   4C 9C 9D   JMP $9D9C    und wieder herstellen

****************************** Aktuelle Koordinaten in Zielkoordinaten

9DF2   A0 00      LDY #$00     Offset auf X-Startkoordinaten
9DF4   20 F9 9D   JSR $9DF9    Koordinate kopieren
9DF7   A0 02      LDY #$02     Offset auf Y-Startkoordinaten

****************************** Eine Koordinate kopieren

9DF9   B9 35 11   LDA $1135,Y  Low-Byte Quelle laden
9DFC   99 31 11   STA $1131,Y  und an Ziel speichern
9DFF   B9 36 11   LDA $1136,Y  High-Byte Quelle laden
9E02   99 32 11   STA $1132,Y  und an Ziel speichern
9E05   60         RTS

****************************** Ausdruck in (A)/(Y) holen

9E06   20 86 03   JSR $0386    CHRGOT
9E09   F0 0C      BEQ $9E17    Wenn Trennzeichen, Skip
9E0B   20 5C 79   JSR $795C    Test auf Komma
9E0E   C9 2C      CMP #$2C     Nächstes Zeichen = ',' ?
9E10   F0 05      BEQ $9E17    Ja, Skip
9E12   20 12 88   JSR $8812    Ausdruck in (A)/(Y) holen
9E15   38         SEC          SEC = Ausdruck gefunden
9E16   60         RTS
9E17   A9 00      LDA #$00     (A)/(Y) = 0
9E19   A8         TAY
9E1A   18         CLC          CLC = Kein Ausdruck gefunden
9E1B   60         RTS

****************************** Byte-Wert in (X) holen

9E1C   A2 00      LDX #$00     Wert mit 0 vorbesetzen
9E1E   20 86 03   JSR $0386    CHRGOT
9E21   F0 F7      BEQ $9E1A    Wenn Trennzeichen, Ende
```

```
9E23   20 5C 79   JSR $795C     Test auf Komma
9E26   C9 2C      CMP #$2C      Nächstes Zeichen = ',' ?
9E28   F0 F0      BEQ $9E1A     Ja, Ende
9E2A   20 F4 87   JSR $87F4     Byte-Wert in (X) holen
9E2D   38         SEC           SEC = Wert gefunden
9E2E   60         RTS

****************************** Farbquelle auswerten

9E2F   20 74 A0   JSR $A074     Test auf HIRES-Bereich
9E32   A2 01      LDX #$01      Wert mit 1 vorbesetzen
9E34   20 86 03   JSR $0386     CHRGOT
9E37   F0 13      BEQ $9E4C     Wenn Trennzeichen, Skip
9E39   C9 2C      CMP #$2C      ',' ?
9E3B   F0 0F      BEQ $9E4C     Ja, Skip
9E3D   20 F4 87   JSR $87F4     Byte-Wert holen
9E40   E0 04      CPX #$04      > 3 ?
9E42   B0 0B      BCS $9E4F     Ja, Fehler
9E44   E0 02      CPX #$02
9E46   24 D8      BIT $D8       Graphikschirm ?
9E48   30 02      BMI $9E4C     Ja, Skip
9E4A   B0 03      BCS $9E4F     Wenn Zahl > 1 und keine Graphik, Fehler
9E4C   86 83      STX $83       Farbquelle speichern
9E4E   60         RTS
9E4F   4C 28 7D   JMP $7D28     'ILLEGAL QUANTITY ERROR'

****************************** Koordinate holen, keine Angabe erlaubt

9E52   20 86 03   JSR $0386     CHRGOT
9E55   F0 07      BEQ $9E5E     Wenn Trennzeichen, Skip
9E57   20 5C 79   JSR $795C     Test auf Komma
9E5A   C9 2C      CMP #$2C      Nächstes Zeichen = ',' ?
9E5C   D0 12      BNE $9E70     Nein, Skip
9E5E   A0 00      LDY #$00      Offset auf 0 setzen
9E60   B9 31 11   LDA $1131,Y   Aktuelle Koordinate laden
9E63   9D 31 11   STA $1131,X   und an Ziel setzen
9E66   E8         INX           Offset erhöhen
9E67   C8         INY           Offset erhöhen
9E68   C0 04      CPY #$04      Koordinate kopiert ?
9E6A   D0 F4      BNE $9E60     Nein, nochmal
9E6C   60         RTS

****************************** Koordinate holen, Angabe zwingend

9E6D   20 5C 79   JSR $795C     Test auf Komma

9E70   8E 78 11   STX $1178     Koordinatenoffset setzen
9E73   20 08 9F   JSR $9F08     Eine Koordinate holen
```

```
9E76   20 86 03   JSR $0386     CHRGOT
9E79   C9 2C      CMP #$2C      ',' ?
9E7B   F0 56      BEQ $9ED3     Ja, Skip
9E7D   C9 3B      CMP #$3B      ';' ?
9E7F   F0 03      BEQ $9E84     Ja, Skip
9E81   4C 6C 79   JMP $796C     'SYNTAX'
9E84   20 80 03   JSR $0380     CHRGET
9E87   20 12 88   JSR $8812     Ausdruck in (Y)/(A) holen
9E8A   85 77      STA $77       (A) und (Y) vertauschen
9E8C   98         TYA
9E8D   A4 77      LDY $77
9E8F   20 77 9A   JSR $9A77     Winkelwerte für (A)/(Y) berechnen
9E92   AE 78 11   LDX $1178
9E95   BD 31 11   LDA $1131,X   Y-Wert = X-Wert
9E98   9D 33 11   STA $1133,X
9E9B   BD 32 11   LDA $1132,X
9E9E   9D 34 11   STA $1134,X
9EA1   20 4A 9D   JSR $9D4A     Koordinaten skalieren
9EA4   A9 0E      LDA #$0E
9EA6   8D 79 11   STA $1179
9EA9   18         CLC
9EAA   AE 78 11   LDX $1178     Koordinatennummer laden
9EAD   20 CE 9A   JSR $9ACE     Sinus mit Koordinate multiplizieren
9EB0   9D 31 11   STA $1131,X
9EB3   98         TYA
9EB4   9D 32 11   STA $1132,X
9EB7   A0 00      LDY #$00      Offset auf Koordinate laden
9EB9   4E 79 11   LSR $1179     Relativkoordinate ?
9EBC   90 02      BCC $9EC0     Nein, Skip
9EBE   A0 02      LDY #$02      Offset auf Koordinate laden
9EC0   20 6B 9D   JSR $9D6B     Wert aufaddieren
9EC3   9D 31 11   STA $1131,X   Neue Koordinate speichern
9EC6   98         TYA           High-Byte in (A)
9EC7   9D 32 11   STA $1132,X   High-Byte speichern
9ECA   E8         INX           Offset erhöhen
9ECB   E8         INX           Offset erhöhen
9ECC   4E 79 11   LSR $1179     Alles verarbeitet ?
9ECF   D0 DC      BNE $9EAD     Nein, nochmal
9ED1   18         CLC
9ED2   60         RTS

******************************** Koordinate holen und auswerten

9ED3   20 80 03   JSR $0380     CHRGET
9ED6   EE 78 11   INC $1178     Koordinatenoffset auf nächste
9ED9   EE 78 11   INC $1178     Koordinate setzen
9EDC   20 08 9F   JSR $9F08     Eine Koordinate auswerten
9EDF   AE 78 11   LDX $1178     Offset laden
```

```
9EE2   CA         DEX            Offset erniedrigen
9EE3   CA         DEX            Offset erniedrigen
9EE4   20 4A 9D   JSR $9D4A      Koordinaten skalieren
9EE7   A0 02      LDY #$02       Offset auf 2. Koordinate setzen
9EE9   AE 78 11   LDX $1178      Koordinatenoffset laden
9EEC   E8         INX            Offset erhöhen
9EED   E8         INX            Offset erhöhen
9EEE   CA         DEX            Offset erniedrigen
9EEF   CA         DEX            Offset erniedrigen
9EF0   4E 79 11   LSR $1179      Relativkoordinate ?
9EF3   90 0A      BCC $9EFF      Nein, Skip
9EF5   20 6D 9D   JSR $9D6D      Koordinatenwert aufaddieren
9EF8   9D 31 11   STA $1131,X    Koordinatenwert setzen
9EFB   98         TYA            High-Byte in (A) holen
9EFC   9D 32 11   STA $1132,X    High-Byte speichern
9EFF   A0 00      LDY #$00       Offset auf 1. Koordinate setzen
9F01   EC 78 11   CPX $1178      2 Koordinaten verarbeitet ?
9F04   F0 E8      BEQ $9EEE      Nein, weitermachen
9F06   18         CLC
9F07   60         RTS

****************************** 1 Koordinate holen

9F08   20 86 03   JSR $0386      CHRGOT
9F0B   C9 AA      CMP #$AA       Token für '+' (Relativkoordinate) ?
9F0D   F0 05      BEQ $9F14      Ja, Skip
9F0F   C9 AB      CMP #$AB       Token für '-' (Relativkoordinate) ?
9F11   F0 01      BEQ $9F14      Ja, Skip
9F13   18         CLC
9F14   2E 79 11   ROL $1179      Bei '+' oder '-' BIT 0 = 1
9F17   20 12 88   JSR $8812      Koordinatenausdruck auswerten
9F1A   AE 78 11   LDX $1178      Offset laden
9F1D   9D 32 11   STA $1132,X    und aktuelle Koordinate speichern
9F20   98         TYA            High-Byte in (A) holen
9F21   9D 31 11   STA $1131,X    High-Byte speichern
9F24   60         RTS

****************************** Werte für Graphikfunktionen

9F25   FF AA 55 00               Bitmuster für Farbmaskierung

9F29   00 00 2C 71 57 8D 80 00   Sinuswerte für 10 Grad-Abschnitte
9F2D   A4 8F C4 19 DD B2 F0 90   High/Low
9F35   FC 1C FF FF

9F3D   04 72 04 50 04 0B 03 A8   Feinabstimmung innerhalb der 10 Grad
9F45   03 28 02 90 01 E3 01 28   High/Low
```

```
9F4D   00 63

****************************** Falls nötig HIRES-Bereich reservieren

9F4F   A5 76        LDA $76      HIRES-Bereich schon reserviert ?
9F51   F0 01        BEQ $9F54    Nein, Skip
9F53   60           RTS
9F54   AD 11 12     LDA $1211    Programmende-High
9F57   18           CLC
9F58   69 24        ADC #$24     + 9 KByte
9F5A   B0 0E        BCS $9F6A    Wenn > 255, Fehler
9F5C   85 62        STA $62      Wert retten
9F5E   CD 13 12     CMP $1213    Wert > Speicherende ?
9F61   90 0A        BCC $9F6D    Nein, Skip
9F63   D0 05        BNE $9F6A    Ja, Fehler
9F65   CC 12 12     CPY $1212    Wert > Speicherende ?
9F68   90 03        BCC $9F6D    Nein, Skip
9F6A   4C 3A 4D     JMP $4D3A    'OUT OF MEMORY'
9F6D   C6 76        DEC $76      HIRES-Flag auf EIN setzen
9F6F   AD 10 12     LDA $1210    Neues Programmende in ($24)
9F72   85 24        STA $24
9F74   A5 62        LDA $62
9F76   85 25        STA $25
9F78   AE 10 12     LDX $1210    Programmende in ($26)
9F7B   86 26        STX $26
9F7D   AD 11 12     LDA $1211
9F80   85 27        STA $27
9F82   38           SEC
9F83   E9 1C        SBC #$1C     - High-Programmstart
9F85   A8           TAY          (Besser wäre SBC $2E!)
9F86   8A           TXA
9F87   49 FF        EOR #$FF
9F89   85 50        STA $50      ($50)= - Programmlänge
9F8B   98           TYA
9F8C   49 FF        EOR #$FF
9F8E   85 51        STA $51
9F90   A0 00        LDY #$00
9F92   E6 50        INC $50      Endezähler erhöhen
9F94   D0 04        BNE $9F9A
9F96   E6 51        INC $51
9F98   F0 18        BEQ $9FB2    Wenn Ende erreicht, Skip
9F9A   A5 24        LDA $24      Zielzeiger erniedrigen
9F9C   D0 02        BNE $9FA0
9F9E   C6 25        DEC $25
9FA0   C6 24        DEC $24
9FA2   A5 26        LDA $26      Quellzeiger erniedrigen
9FA4   D0 02        BNE $9FA8
9FA6   C6 27        DEC $27
```

```
9FA8   C6 26      DEC $26
9FAA   20 C0 03   JSR $03C0     Programm byteweise übertragen
9FAD   91 24      STA ($24),Y
9FAF   4C 92 9F   JMP $9F92     Zum Schleifenstart
9FB2   18         CLC
9FB3   AD 11 12   LDA $1211     Programmende neu setzen
9FB6   69 24      ADC #$24      + 9 KByte
9FB8   8D 11 12   STA $1211
9FBB   A5 2E      LDA $2E       Programmstart neu setzen
9FBD   69 24      ADC #$24      + 9 KByte
9FBF   85 2E      STA $2E
9FC1   A5 44      LDA $44       DATA-Adresse erhöhen
9FC3   69 24      ADC #$24
9FC5   85 44      STA $44

9FC7   20 4F 4F   JSR $4F4F     Zeilenverkettung korrigieren
9FCA   20 82 4F   JSR $4F82
9FCD   24 7F      BIT $7F       Direktmodus ?
9FCF   10 2D      BPL $9FFE     Ja, Skip
9FD1   A2 24      LDX #$24      + 9 KByte
9FD3   24 76      BIT $76       HIRES-Bereich setzen ?
9FD5   30 02      BMI $9FD9     Ja, Skip
9FD7   A2 DC      LDX #$DC      - 9 KByte
9FD9   8A         TXA
9FDA   18         CLC
9FDB   65 3E      ADC $3E       PC neu setzen
9FDD   85 3E      STA $3E
9FDF   8A         TXA
9FE0   18         CLC
9FE1   6D 03 12   ADC $1203     CONT-Zeiger neu setzen
9FE4   8D 03 12   STA $1203
9FE7   8A         TXA
9FE8   18         CLC
9FE9   6D 0F 12   ADC $120F     Fehlerzeiger neu setzen
9FEC   8D 0F 12   STA $120F
9FEF   20 47 50   JSR $5047     BASIC-Stackpointer in Hilfszeiger
9FF2   A5 3F      LDA $3F       Stack leer ?
9FF4   C9 FF      CMP #$FF
9FF6   D0 07      BNE $9FFF     Nein, Skip
9FF8   A5 40      LDA $40
9FFA   C9 09      CMP #$09      Stack leer ?
9FFC   D0 01      BNE $9FFF     Nein, Skip
9FFE   60         RTS
9FFF   A0 00      LDY #$00      Offset auf 0 setzen
```

```
A001   B1 3F      LDA ($3F),Y  Zeichen aus Stack
A003   C9 81      CMP #$81     Token für FOR
A005   D0 09      BNE $A010    Nein, Skip
A007   A0 10      LDY #$10     Offset laden
A009   20 62 A0   JSR $A062    PC auf Stack korrigieren
A00C   A9 12      LDA #$12     Offset auf nächstes Stackelement laden
A00E   D0 07      BNE $A017    Unbedingter Sprung
A010   A0 04      LDY #$04     Offset laden
A012   20 62 A0   JSR $A062    PC auf Stack korrigieren
A015   A9 05      LDA #$05     Offset auf nächstes Stackelement laden
A017   18         CLC
A018   65 3F      ADC $3F      Hilfsstackpointer erhöhen
A01A   85 3F      STA $3F
A01C   90 D4      BCC $9FF2    Kein Übertrag, Skip
A01E   E6 40      INC $40      Übertrag berücksichtigen
A020   D0 D0      BNE $9FF2    Unbedingter Sprung zum Schleifenstart

****************************** HIRES-Bereich löschen

A022   A5 76      LDA $76      HIRES-Bereich reserviert ?
A024   D0 01      BNE $A027    Ja, Skip
A026   60         RTS
A027   A0 00      LDY #$00     Offset auf 0 setzen
A029   84 76      STY $76      HIRES-Flag löschen
A02B   84 24      STY $24      Low-Bytes der Pointer löschen
A02D   84 26      STY $26
A02F   A9 1C      LDA #$1C     High-Byte des Zielpointers setzen
A031   85 25      STA $25
A033   A9 40      LDA #$40     High-Byte des Quellpointers setzen
A035   85 27      STA $27
A037   20 C0 03   JSR $03C0    Programm byteweise übertragen
A03A   91 24      STA ($24),Y  Programmbyte speichern
A03C   C8         INY          Block fertig übertragen ?
A03D   D0 F8      BNE $A037    Nein, weitermachen
A03F   E6 25      INC $25      High-Byte Ziel erhöhen
A041   E6 27      INC $27      High-Byte Quelle erhöhen
A043   AD 11 12   LDA $1211    Programmende erreicht ?
A046   C5 27      CMP $27
A048   B0 ED      BCS $A037    Nein, weiterschleifen
A04A   38         SEC
A04B   A5 2E      LDA $2E      Programmstart korrigieren
A04D   E9 24      SBC #$24     - 9 KByte
A04F   85 2E      STA $2E
A051   AD 11 12   LDA $1211    Programmende korrigieren
A054   E9 24      SBC #$24     - 9 KByte
A056   8D 11 12   STA $1211
A059   A5 44      LDA $44      DATA-Zeiger korrigieren
A05B   E9 24      SBC #$24     - 9 KByte
```

```
A05D   85 44      STA $44
A05F   4C C7 9F   JMP $9FC7     Rest der Zeiger neu setzen

****************************** PC auf BASIC-Stack korrigieren

A062   B1 3F      LDA ($3F),Y   High-Byte laden
A064   24 76      BIT $76       HIRES-Bereich setzen ?
A066   D0 06      BNE $A06E     Ja, Skip
A068   38         SEC
A069   E9 24      SBC #$24      - 9 KByte
A06B   91 3F      STA ($3F),Y
A06D   60         RTS
A06E   18         CLC
A06F   69 24      ADC #$24      + 9 KByte
A071   91 3F      STA ($3F),Y
A073   60         RTS

****************************** Test auf HIRES-Bereich

A074   A5 76      LDA $76       HIRES-Bereich reserviert ?
A076   F0 01      BEQ $A079     Nein, Fehler
A078   60         RTS
A079   A2 23      LDX #$23      'NO GRAPHICS AREA'
A07B   4C 3C 4D   JMP $4D3C     Fehler ausgeben

****************************** BASIC-Befehl DIRECTORY/CATALOG

A07E   20 BF A3   JSR $A3BF     Parameter auswerten
A081   A5 80      LDA $80       Flag für Parameter laden
A083   29 E6      AND #$E6      Parameter korrekt angegeben ?
A085   F0 03      BEQ $A08A     Ja, Skip
A087   4C 6C 79   JMP $796C     'SYNTAX'
A08A   A0 01      LDY #$01      Offset auf Befehlstabelle laden
A08C   A2 01      LDX #$01      Stringlänge vorbesetzen
A08E   A5 80      LDA $80       Flag für Parameter laden
A090   29 11      AND #$11      Drive oder Filename angegeben ?
A092   F0 06      BEQ $A09A     Nein, Skip
A094   4A         LSR           Filename ?
A095   90 02      BCC $A099     Nein, Skip
A097   E8         INX           Stringlänge anpassen
A098   E8         INX
A099   E8         INX
A09A   8A         TXA
A09B   20 67 A6   JSR $A667     Diskstring zusammenstellen
A09E   A9 00      LDA #$00
A0A0   AA         TAX           Bank = 0
A0A1   20 87 92   JSR $9287     Bank 0 für Diskoperation setzen
A0A4   A0 60      LDY #$60      Sekundäradresse =0 (LOAD)
```

```
A0A6  AE 1C 01  LDX $011C   Gerätenummer laden
A0A9  A9 00     LDA #$00    Filenummer auf 0
A0AB  20 57 92  JSR $9257   Fileparameter setzen
A0AE  38        SEC
A0AF  20 D8 90  JSR $90D8   BASIC-OPEN
A0B2  90 09     BCC $A0BD   Wenn fehlerfrei, Skip
A0B4  48        PHA         Fehler retten
A0B5  20 14 A1  JSR $A114   File wieder schließen
A0B8  68        PLA         Fehler wieder holen
A0B9  AA        TAX         Fehlernummer in (X)
A0BA  4C 3C 4D  JMP $4D3C   und Fehlermeldung ausgeben

A0BD  A2 00     LDX #$00    Filenummer 0
A0BF  20 45 A8  JSR $A845   ROMs einschalten
A0C2  20 C6 FF  JSR $FFC6   Eingabegerät setzen
A0C5  A0 03     LDY #$03    Startadresse und Linkadresse
A0C7  8C 74 11  STY $1174   überlesen
A0CA  20 63 92  JSR $9263   Wert-Low lesen
A0CD  8D 75 11  STA $1175   und setzen
A0D0  20 51 92  JSR $9251   Status holen
A0D3  D0 3F     BNE $A114   Wenn gesetzt, CLOSE und Ende
A0D5  20 63 92  JSR $9263   Wert-High lesen
A0D8  8D 76 11  STA $1176   und setzen
A0DB  20 51 92  JSR $9251   Status holen
A0DE  D0 34     BNE $A114   Wenn gesetzt, CLOSE und Ende
A0E0  CE 74 11  DEC $1174   Noch eine Adresse überlesen ?
A0E3  D0 E5     BNE $A0CA   Ja, zum Schleifenstart
A0E5  AE 75 11  LDX $1175   Blockanzahl (Zeilennummer) laden
A0E8  AD 76 11  LDA $1176
A0EB  20 32 8E  JSR $8E32   Und ausgeben
A0EE  A9 20     LDA #$20    ' '
A0F0  20 69 92  JSR $9269   Ausgeben
A0F3  20 63 92  JSR $9263   Zeichen holen
A0F6  48        PHA         Und retten
A0F7  20 51 92  JSR $9251   Status holen
A0FA  D0 17     BNE $A113   Wenn gesetzt, CLOSE und Ende
A0FC  68        PLA         Zeichen wieder holen
A0FD  F0 06     BEQ $A105   Wenn Zeilenende, Skip
A0FF  20 69 92  JSR $9269   Zeichen ausgeben
A102  4C F3 A0  JMP $A0F3   Und nächstes Zeichen holen
A105  A9 0D     LDA #$0D    (CR)
A107  20 69 92  JSR $9269   Ausgeben
A10A  20 93 92  JSR $9293   Test auf STOP-Taste
A10D  F0 05     BEQ $A114   Wenn STOP, CLOSE und Ende
A10F  A0 02     LDY #$02    Linkadresse überlesen
A111  D0 B4     BNE $A0C7   Und zum Schleifenstart
A113  68        PLA         Zeichen wieder holen
A114  20 6F 92  JSR $926F   CLRCH I/O rücksetzen
```

```
A117   A9 00      LDA #$00     Filenummer 0 laden
A119   18         CLC
A11A   4C 75 92   JMP $9275    CLOSE File schliessen

******************************** BASIC-Befehl DOPEN

A11D   A9 22      LDA #$22     Bits für verbotene Angaben laden
A11F   20 C1 A3   JSR $A3C1    Parameter auswerten
A122   20 6F A7   JSR $A76F    Syntaxprüfung
A125   20 57 A1   JSR $A157    Freie Sekundäradresse suchen
A128   A0 05      LDY #$05     Offset auf Befehlstabelle laden
A12A   A2 04      LDX #$04     Stringlänge laden
A12C   24 80      BIT $80      A12E   50 13      BVC $A143
A130   A2 08      LDX #$08     Stringlänge korrigieren
A132   D0 0F      BNE $A143

******************************** BASIC-Befehl APPEND

A134   A9 E2      LDA #$E2     Bits für verbotene Angaben laden
A136   20 C1 A3   JSR $A3C1    Parameter auswerten
A139   20 6F A7   JSR $A76F    Syntaxprüfung
A13C   20 57 A1   JSR $A157    Freie Sekundäradresse suchen
A13F   A0 16      LDY #$16     Offset auf Befehlstabelle laden
A141   A2 05      LDX #$05     Stringlänge laden

A143   8A         TXA
A144   20 67 A6   JSR $A667    Diskstring zusammenstellen
A147   20 6F 92   JSR $926F    CLRCH I/O rücksetzen
A14A   A9 00      LDA #$00     Filenummer 0 laden
A14C   AA         TAX          Banknummer setzen
A14D   20 87 92   JSR $9287    Bank auf 0 setzen
A150   20 D8 90   JSR $90D8    Filenummer 0 OPEN
A153   38         SEC
A154   4C 75 92   JMP $9275    CLOSE

******************************** Freie Sekundäradresse suchen

A157   A0 61      LDY #$61     Sekundäradresse 1 laden
A159   C8         INY          Adresse erhöhen
A15A   C0 6F      CPY #$6F     Sekundäradresse 15 schon erreicht ?
A15C   F0 0C      BEQ $A16A    Ja, Fehler
A15E   20 45 A8   JSR $A845    ROMs einschalten
A161   20 5C FF   JSR $FF5C    Sekundäradresse in Tabelle suchen
A164   90 F3      BCC $A159    Wenn gefunden, weitersuchen
A166   8C 1D 01   STY $011D    Sekundäradresse setzen
A169   60         RTS
A16A   A2 01      LDX #$01     'TOO MANY FILES'
```

```
A16C   4C 3C 4D   JMP $4D3C    Fehler ausgeben

****************************** BASIC-Befehl DCLOSE

A16F   A9 F3      LDA #$F3     Bits für verbotene Angaben laden
A171   20 C1 A3   JSR $A3C1    Parameter auswerten
A174   20 0D A8   JSR $A80D    DS$ löschen
A177   A5 80      LDA $80      Flag für Parameter laden
A179   29 04      AND #$04     Filenummer angegeben ?
A17B   F0 06      BEQ $A183    Nein, Skip
A17D   AD 1B 01   LDA $011B    Filenummer laden
A180   4C 75 92   JMP $9275    Und File schließen
A183   AD 1C 01   LDA $011C    Gerätenummer laden
A186   20 45 A8   JSR $A845    ROMs einschalten
A189   4C 4A FF   JMP $FF4A    Alle Files auf aktuellem Gerät schließen

****************************** BASIC-Befehl DSAVE

A18C   A9 66      LDA #$66     Bits für verbotene Angaben laden
A18E   20 C1 A3   JSR $A3C1    Parameter auswerten
A191   20 50 A7   JSR $A750    Syntaxprüfung
A194   A0 05      LDY #$05     Offset auf Befehlstabelle setzen
A196   A9 04      LDA #$04     Stringlänge setzen
A198   20 67 A6   JSR $A667    Diskstring zusammenstellen
A19B   A9 00      LDA #$00     Filenummer 0
A19D   AA         TAX          Bank 0
A19E   20 87 92   JSR $9287    Bank 0 für Diskoperationen setzen
A1A1   4C 15 91   JMP $9115    SAVE-Befehl aufrufen

****************************** BASIC-Befehl DVERIFY

A1A4   A9 01      LDA #$01     Flag für Verify
A1A6   2C         .BYTE $2C    Nächsten Befehl überlesen

****************************** BASIC-Befehl DLOAD

A1A7   A9 00      LDA #$00     Flag für Load
A1A9   85 0C      STA $0C      Flag setzen
A1AB   A9 E6      LDA #$E6     Bits für verbotene Parameter setzen
A1AD   20 C1 A3   JSR $A3C1    Parameter holen
A1B0   20 50 A7   JSR $A750    Syntaxprüfung
A1B3   A9 00      LDA #$00     Logische Adresse auf 0 setzen
A1B5   8D 1D 01   STA $011D
A1B8   A0 05      LDY #$05     Offset auf Befehlstabelle laden
A1BA   A9 04      LDA #$04     Stringlänge setzen
A1BC   20 67 A6   JSR $A667    Diskstring zusammenstellen
A1BF   A9 00      LDA #$00
A1C1   AA         TAX          Bank 0 setzen
```

```
A1C2   20 87 92   JSR $9287   Bank 0 für Diskoperationen setzen
A1C5   4C 33 91   JMP $9133   LOAD-Befehl aufrufen

****************************** BASIC-Befehl BSAVE

A1C8   A9 66      LDA #$66    Bits für verbotene Parameter setzen
A1CA   A2 F8      LDX #$F8
A1CC   20 C3 A3   JSR $A3C3   Parameter holen
A1CF   20 50 A7   JSR $A750   Syntaxprüfung
A1D2   A5 81      LDA $81     Flag für Adressen laden
A1D4   29 06      AND #$06
A1D6   C9 06      CMP #$06    Start- und Endadresse angegeben ?
A1D8   F0 03      BEQ $A1DD   Ja, Skip
A1DA   4C 6C 79   JMP $796C   'SYNTAX'
A1DD   AD 1A 01   LDA $011A   Endadresse =< Startadresse ?
A1E0   CD 18 01   CMP $0118
A1E3   90 30      BCC $A215   Ja, Fehler
A1E5   D0 0A      BNE $A1F1   Nein, Skip
A1E7   AD 19 01   LDA $0119   Endadresse =< Startadresse ?
A1EA   CD 17 01   CMP $0117
A1ED   90 26      BCC $A215   Ja, Fehler
A1EF   F0 24      BEQ $A215   Ja, Fehler
A1F1   A0 05      LDY #$05    Offset auf Befehlstabelle laden
A1F3   A9 04      LDA #$04    Stringlänge laden
A1F5   20 67 A6   JSR $A667   Diskstring zusammenstellen
A1F8   AD 1F 01   LDA $011F   Sekundäradresse laden
A1FB   A2 00      LDX #$00    Bank 0
A1FD   20 87 92   JSR $9287   Bank 0 für Diskoperationen setzen
A200   AE 17 01   LDX $0117   Startadresse in ($5A) kopieren
A203   AC 18 01   LDY $0118
A206   A9 5A      LDA #$5A    Flag für Startadresse in ($5A)
A208   86 5A      STX $5A     Startadresse vorbesetzen
A20A   84 5B      STY $5B
A20C   AE 19 01   LDX $0119   Endadresse laden
A20F   AC 1A 01   LDY $011A
A212   4C 1D 91   JMP $911D   SAVE-Befehl aufrufen
A215   4C 28 7D   JMP $7D28   'ILLEGAL QUANTITY'

****************************** BASIC-Befehl BLOAD

A218   A9 E6      LDA #$E6    Bits für verbotene Parameter setzen
A21A   A2 FC      LDX #$FC
A21C   20 C3 A3   JSR $A3C3   Parameter holen
A21F   20 50 A7   JSR $A750   Syntaxprüfung
A222   AE 17 01   LDX $0117   Startadresse holen
A225   AC 18 01   LDY $0118
A228   A9 00      LDA #$00    Sekundäradresse 0
A22A   E0 FF      CPX #$FF    Startadresse angegeben ?
```

```
A22C   D0 06      BNE $A234    Ja, Skip
A22E   C0 FF      CPY #$FF
A230   D0 02      BNE $A234    Ja, Skip
A232   A9 FF      LDA #$FF     Sekundäradresse 1
A234   8D 1D 01   STA $011D    Sekundäradresse setzen
A237   A0 05      LDY #$05     Offset auf Befehlstabelle laden
A239   A9 04      LDA #$04     Stringlänge setzen
A23B   20 67 A6   JSR $A667    Diskstring zusammenstellen
A23E   AD 1F 01   LDA $011F    Sekundäradresse laden
A241   A2 00      LDX #$00     Bank 0
A243   20 87 92   JSR $9287    Bank 0 für Diskoperationen setzen
A246   A9 00      LDA #$00     Load-Flag
A248   AE 17 01   LDX $0117    Startadresse laden
A24B   AC 18 01   LDY $0118
A24E   20 D5 FF   JSR $FFD5    LOADSP Load-Routine
A251   08         PHP          Status retten
A252   20 43 92   JSR $9243    DS$ für ungültig erklären
A255   28         PLP          Status wieder holen
A256   90 03      BCC $A25B    Wenn fehlerfrei, Skip
A258   4C D0 90   JMP $90D0    Fehlerauswertung
A25B   20 51 92   JSR $9251    Status holen
A25E   29 BF      AND #$BF     EOF ?
A260   F0 03      BEQ $A265    Nein, Skip
A262   4C 67 91   JMP $9167    LOAD-Befehl aufrufen
A265   18         CLC
A266   60         RTS

******************************** BASIC-Befehl HEADER

A267   20 BF A3   JSR $A3BF    Parameter holen
A26A   20 49 A7   JSR $A749    Syntaxprüfung
A26D   29 01      AND #$01     Diskettenname angegeben ?
A26F   C9 01      CMP #$01
A271   D0 61      BNE $A2D4    Nein, Fehler
A273   20 7B 92   JSR $927B    CLALL
A276   20 E1 A7   JSR $A7E1    'ARE YOU SURE?'
A279   D0 25      BNE $A2A0    Nein, nicht sicher, Ende
A27B   A0 1B      LDY #$1B     Offset auf Befehlstabelle laden
A27D   A9 04      LDA #$04     Stringlänge laden
A27F   AE 20 01   LDX $0120    ID angegeben ?
A282   F0 02      BEQ $A286    Nein, Skip
A284   A9 06      LDA #$06     Stringlänge korrigieren
A286   20 97 A3   JSR $A397    String zusammenstellen und übergeben
A289   20 78 A7   JSR $A778    DS$ holen
A28C   24 7F      BIT $7F      Direktmodus ?
A28E   30 10      BMI $A2A0    Nein, Ende
A290   A0 00      LDY #$00     Offset auf 0
A292   A9 7B      LDA #$7B     Zeiger auf Deskriptor von DS$
```

```
A294   20 AB 03   JSR $03AB    Erstes Zeichen aus DS$ laden
A297   C9 32      CMP #$32     Fehlernummer < 20 ?
A299   90 05      BCC $A2A0    Ja, Skip
A29B   A2 24      LDX #$24     'BAD DISK'
A29D   4C 3C 4D   JMP $4D3C    Fehler ausgeben
A2A0   60         RTS

******************************** BASIC-Befehl SCRATCH

A2A1   20 BF A3   JSR $A3BF    Parameter holen
A2A4   20 49 A7   JSR $A749    Syntaxprüfung
A2A7   20 E1 A7   JSR $A7E1    'ARE YOU SURE?'
A2AA   D0 27      BNE $A2D3    Nein, nicht sicher, Ende
A2AC   A0 37      LDY #$37     Offset auf Befehlstabelle laden
A2AE   A9 04      LDA #$04     Stringlänge setzen
A2B0   20 97 A3   JSR $A397    String zusammenstellen und übergeben
A2B3   20 78 A7   JSR $A778    DS$ holen
A2B6   24 7F      BIT $7F      Direktmodus ?
A2B8   30 19      BMI $A2D3    Nein, Skip
A2BA   A9 0D      LDA #$0D     (CR)
A2BC   20 69 92   JSR $9269    Ausgeben
A2BF   A0 00      LDY #$00     Offset auf 0
A2C1   A9 7B      LDA #$7B     Zeiger auf Deskriptor von DS$
A2C3   20 AB 03   JSR $03AB    Zeichen aus DS$ holen
A2C6   F0 06      BEQ $A2CE    Wenn 0, Ende
A2C8   20 69 92   JSR $9269    Zeichen ausgeben
A2CB   C8         INY          Offset erhöhen
A2CC   D0 F3      BNE $A2C1    Und weitermachen
A2CE   A9 0D      LDA #$0D     (CR)
A2D0   20 DF 90   JSR $90DF    Ausgeben
A2D3   60         RTS
A2D4   4C 6C 79   JMP $796C    'SYNTAX'

******************************** BASIC-Befehl RECORD

A2D7   A9 23      LDA #$23     '#'
A2D9   20 5E 79   JSR $795E    Test auf Code
A2DC   20 F4 87   JSR $87F4    Filenummer holen
A2DF   E0 00      CPX #$00     Filenummer = 0 ?
A2E1   F0 37      BEQ $A31A    Ja, Fehler
A2E3   8E 1B 01   STX $011B    Filenummer setzen
A2E6   20 0F 88   JSR $880F    Datensatznummer holen
A2E9   A2 01      LDX #$01     Wert für erstes Byte vorbesetzen
A2EB   20 1E 9E   JSR $9E1E    Byte-Nummer holen
A2EE   E0 00      CPX #$00     Bytenummer = 0 ?
A2F0   F0 28      BEQ $A31A    Ja, Fehler
A2F2   E0 FF      CPX #$FF     Bytenummer = 255 ?
A2F4   F0 24      BEQ $A31A    Ja, Fehler
```

```
A2F6   8E 1E 01   STX $011E    Bytenummer setzen
A2F9   AD 1B 01   LDA $011B    Filenummer laden
A2FC   20 45 A8   JSR $A845    ROMs einschalten
A2FF   20 59 FF   JSR $FF59    LKUPLA Filenummer suchen
A302   B0 19      BCS $A31D    Wenn nicht gefunden, Fehler
A304   8C ED 11   STY $11ED    Sekundäradresse setzen
A307   8E 1C 01   STX $011C    Geräteadresse setzen
A30A   A9 00      LDA #$00     Filenummer 0 setzen
A30C   8D 1B 01   STA $011B
A30F   A9 6F      LDA #$6F     Sekundäradresse setzen
A311   8D 1D 01   STA $011D
A314   A0 3B      LDY #$3B     Offset auf Befehlstabelle laden
A316   A9 04      LDA #$04     Stringlänge laden
A318   D0 7D      BNE $A397    Unbedingter Sprung
A31A   4C 28 7D   JMP $7D28    'ILLEGAL QUANTITY'
A31D   A2 04      LDX #$04     'FILE NOT FOUND'
A31F   4C 3C 4D   JMP $4D3C    Fehler ausgeben

****************************** BASIC-Befehl DCLEAR

A322   20 BF A3   JSR $A3BF    Parameter holen
A325   A0 FF      LDY #$FF     Offset auf Befehlstabelle laden
A327   A9 02      LDA #$02     Stringlänge auf 2 setzen
A329   20 97 A3   JSR $A397    String zusammenstellen und übergeben
A32C   4C 83 A1   JMP $A183    Files schließen

****************************** BASIC-Befehl COLLECT

A32F   20 BF A3   JSR $A3BF    Parameter holen
A332   20 5B A7   JSR $A75B    Syntaxprüfung
A335   20 7B 92   JSR $927B    CLALL Offene Dateien löschen
A338   A0 21      LDY #$21     Offset auf Befehlstabelle laden
A33A   A2 01      LDX #$01     Stringlänge setzen
A33C   A5 80      LDA $80      Flag für Parameter laden
A33E   29 10      AND #$10     Laufwerksnummer angegeben ?
A340   F0 01      BEQ $A343    Nein, Skip
A342   E8         INX          Länge auf 2 Zeichen erhöhen
A343   8A         TXA          Stringlänge in (A)
A344   D0 51      BNE $A397    Unbedingter Sprung

****************************** BASIC-Befehl COPY

A346   20 BF A3   JSR $A3BF    Parameter holen
A349   29 30      AND #$30
A34B   C9 30      CMP #$30     Zwei Laufwerksnummern angegeben ?
A34D   D0 06      BNE $A355    Nein, Skip
A34F   A5 80      LDA $80      Flag für Parameter laden
A351   29 C7      AND #$C7     Parameter uneingeschränkt gültig ?
```

```
A353   F0 07      BEQ $A35C   Ja, Skip
A355   A5 80      LDA $80     Flag für Parameter laden
A357   20 60 A7   JSR $A760   Syntaxprüfung
A35A   A5 80      LDA $80     Flag für Parameter laden
A35C   A0 27      LDY #$27    Offset auf Befehlstabelle laden
A35E   A9 08      LDA #$08    Stringlänge setzen
A360   D0 35      BNE $A397   Unbedingter Sprung

******************************* BASIC-Befehl CONCAT

A362   20 BF A3   JSR $A3BF   Parameter holen
A365   20 60 A7   JSR $A760   Syntaxprüfung
A368   A0 0D      LDY #$0D    Offset auf Befehlstabelle laden
A36A   A9 0C      LDA #$0C    Stringlänge setzen
A36C   D0 29      BNE $A397   Unbedingter Sprung

******************************* BASIC-Befehl RENAME

A36E   A9 E4      LDA #$E4    Bits für verbotene Parameter setzen
A370   20 C1 A3   JSR $A3C1   Parameter holen
A373   20 66 A7   JSR $A766   Syntaxprüfung
A376   A0 2F      LDY #$2F    Offset auf Befehlstabelle setzen
A378   A9 08      LDA #$08    Stringlänge setzen
A37A   D0 1B      BNE $A397   Unbedingter Sprung

******************************* BASIC-Befehl BACKUP

A37C   A9 C7      LDA #$C7    Bits für verbotene Parameter setzen
A37E   20 C1 A3   JSR $A3C1   Parameter holen
A381   29 30      AND #$30
A383   C9 30      CMP #$30    Zwei Laufwerksnummern angegeben ?
A385   F0 03      BEQ $A38A   Ja, Skip
A387   4C 6C 79   JMP $796C   'SYNTAX'
A38A   20 E1 A7   JSR $A7E1   'ARE YOU SURE?'
A38D   F0 01      BEQ $A390   Ja, sicher, Skip
A38F   60         RTS
A390   20 83 A1   JSR $A183   Files schließen
A393   A0 23      LDY #$23    Offset auf Befehlstabelle laden
A395   A9 04      LDA #$04    Stringlänge setzen

******************************* Diskstring zusammenstellen und übergeben

A397   20 67 A6   JSR $A667   Diskstring zusammenstellen
A39A   20 6F 92   JSR $926F   CLRCH I/O rücksetzen
A39D   A9 00      LDA #$00
A39F   AA         TAX         Bank 0 setzen
A3A0   20 87 92   JSR $9287   Bank 0 für Diskoperationen setzen
A3A3   38         SEC
```

```
A3A4   20 D8 90   JSR $90D8    BASIC-Open
A3A7   08         PHP          Status retten
A3A8   48         PHA          Fehlernummer retten
A3A9   AD 1B 01   LDA $011B    Filenummer laden
A3AC   38         SEC
A3AD   20 75 92   JSR $9275    CLOSE File schließen
A3B0   68         PLA          Fehlernummer wieder holen
A3B1   28         PLP          Status holen
A3B2   B0 01      BCS $A3B5    Wenn Fehler, zur Auswertung
A3B4   60         RTS
A3B5   4C D0 90   JMP $90D0    Fehlerauswertung

******************************** Vorbesetzung für Adressen
                                 und Fileparameter

A3B8   FF FF FF FF 00 08 6F

******************************** Parameter auswerten

A3BF   A9 00      LDA #$00     Flag für alle Parameter erlaubt

A3C1   A2 FF      LDX #$FF     Flag für keine Adreßangaben erlaubt

A3C3   48         PHA          Beide Flagwerte retten
A3C4   8A         TXA
A3C5   48         PHA
A3C6   A9 00      LDA #$00
A3C8   85 80      STA $80      Parameterstatus löschen
A3CA   85 81      STA $81      Adreßangabestatus löschen
A3CC   A2 22      LDX #$22     Offset laden
A3CE   9D 00 01   STA $0100,X  DOS-Puffer löschen
A3D1   CA         DEX          Puffer gelöscht ?
A3D2   D0 FA      BNE $A3CE    Nein, nochmal
A3D4   A2 06      LDX #$06     Adressen und Fileparameter vorbesetzen
A3D6   BD B8 A3   LDA $A3B8,X  Parameter aus Tabelle laden
A3D9   9D 17 01   STA $0117,X  und speichern
A3DC   CA         DEX          Alle Parameter gesetzt ?
A3DD   10 F7      BPL $A3D6    Nein, nochmal
A3DF   AE D5 03   LDX $03D5    Aktuelle Bank laden
A3E2   8E 1F 01   STX $011F    und setzen
A3E5   20 86 03   JSR $0386    CHRGOT
A3E8   D0 0E      BNE $A3F8    Wenn kein Trennzeichen, Skip
A3EA   68         PLA          Adreßwertflag holen
A3EB   25 81      AND $81      Falsche Werte angegeben ?
A3ED   D0 6B      BNE $A45A    Ja, Fehler
A3EF   68         PLA          Paramterflag holen
A3F0   20 1D A6   JSR $A61D    Status prüfen
A3F3   A5 80      LDA $80      Statuswerte laden
```

```
A3F5   A6 81      LDX $81
A3F7   60         RTS

A3F8   C9 23      CMP #$23     '#' Filenummer ?
A3FA   F0 4B      BEQ $A447    Ja, Skip
A3FC   C9 57      CMP #$57     'W' Writeangabe ?
A3FE   F0 5D      BEQ $A45D    Ja, Skip
A400   C9 4C      CMP #$4C     'L' Satzlänge ?
A402   F0 59      BEQ $A45D    Ja, Skip
A404   C9 52      CMP #$52     'R' Readangabe ?
A406   F0 29      BEQ $A431    Ja, Skip
A408   C9 44      CMP #$44     'D' Driveangabe ?
A40A   F0 73      BEQ $A47F    Ja, Skip
A40C   C9 91      CMP #$91     Token für ON ?
A40E   F0 27      BEQ $A437    Ja, Skip
A410   C9 42      CMP #$42     'B' Bankangabe ?
A412   F0 2E      BEQ $A442    Ja, Skip
A414   C9 55      CMP #$55     'U' Unitangabe ?
A416   F0 25      BEQ $A43D    Ja, Skip
A418   C9 50      CMP #$50     'P' Adreßangabe ?
A41A   D0 03      BNE $A41F    Nein, Skip
A41C   4C B4 A4   JMP $A4B4    Adresse auswerten

A41F   C9 49      CMP #$49     'I' ID Angabe ?
A421   F0 75      BEQ $A498    Ja, Skip
A423   C9 22      CMP #$22     '"' Filename ?
A425   F0 07      BEQ $A42E    Ja, Skip
A427   C9 28      CMP #$28     '(' Filenamenausdruck ?
A429   F0 03      BEQ $A42E    Ja, Skip
A42B   4C 6C 79   JMP $796C    'SYNTAX'
A42E   4C DC A4   JMP $A4DC    Filename übernehmen

A431   20 80 03   JSR $0380    'R' überlesen
A434   4C FB A4   JMP $A4FB    und weiter testen

A437   20 82 A5   JSR $A582    Routine für ON aufrufen
A43A   4C F7 A4   JMP $A4F7    und weiter testen

A43D   20 8D A5   JSR $A58D    Routine für 'U' aufrufen
A440   D0 F8      BNE $A43A    und weiter testen

A442   20 9E A5   JSR $A59E    Banknummer einlesen
A445   F0 F3      BEQ $A43A    und weiter testen

A447   A9 04      LDA #$04     Filenummer schon gelesen ?
A449   20 1D A6   JSR $A61D    Wenn ja, Fehler
A44C   20 F2 A5   JSR $A5F2    Filenummer holen
A44F   E0 00      CPX #$00     Filenummer = 0 ?
```

```
A451   F0 42      BEQ $A495   Ja, Fehler
A453   8E 1B 01   STX $011B   Filenummer setzen
A456   A9 04      LDA #$04    Status für Filenummer gesetzt laden
A458   D0 E0      BNE $A43A   Unbedingter Sprung
A45A   4C 6C 79   JMP $796C   'SYNTAX'

A45D   AA         TAX         'W' oder 'L' retten
A45E   A9 40      LDA #$40    Status für 'W' oder 'L' schon gesetzt ?
A460   20 1D A6   JSR $A61D   Status testen
A463   E0 57      CPX #$57    'W' Writeangabe ?
A465   D0 06      BNE $A46D   Nein, Skip
A467   20 80 03   JSR $0380   'W' überlesen
A46A   4C 7B A4   JMP $A47B   Status setzen
A46D   20 F2 A5   JSR $A5F2   Satzlänge auswerten
A470   E0 00      CPX #$00    Länge = 0 ?
A472   F0 21      BEQ $A495   Ja, Fehler
A474   E0 FF      CPX #$FF    Länge = 255 ?
A476   F0 1D      BEQ $A495   Ja, Fehler
A478   8E 1E 01   STX $011E   Satzlänge setzen
A47B   A9 40      LDA #$40    Status setzen
A47D   D0 14      BNE $A493   Unbedingter Sprung

A47F   A9 10      LDA #$10    Status für Drive 1 schon angegeben
A481   20 1D A6   JSR $A61D   Status testen
A484   20 F2 A5   JSR $A5F2   Drivenummer 1 holen
A487   E0 02      CPX #$02    Zu groß ?
A489   B0 0A      BCS $A495   Ja, Fehler
A48B   8E 12 01   STX $0112   Drivenummer 1 setzen
A48E   8E 14 01   STX $0114   Drivenummer 2 vorbesetzen
A491   A9 10      LDA #$10    Status setzen
A493   D0 62      BNE $A4F7   Unbedingter Sprung
A495   4C 28 7D   JMP $7D28   'ILLEGAL QUANTITY'

A498   AD 22 01   LDA $0122   ID schon angegeben ?
A49B   D0 BD      BNE $A45A   Ja, Fehler
A49D   20 80 03   JSR $0380   Erstes Zeichen der ID holen
A4A0   8D 20 01   STA $0120   und setzen
A4A3   20 80 03   JSR $0380   Zweites Zeichen der ID holen
A4A6   8D 21 01   STA $0121   und setzen
A4A9   A9 FF      LDA #$FF    Flag für ID gelesen setzen
A4AB   8D 22 01   STA $0122
A4AE   20 80 03   JSR $0380   Nächstes Zeichen lesen
A4B1   4C FB A4   JMP $A4FB   und weiter testen

A4B4   A9 02      LDA #$02    Adresse 1 schon angegeben ?
A4B6   20 22 A6   JSR $A622   Wenn ja, Fehler
A4B9   20 05 A6   JSR $A605   Adreßwert holen
A4BC   8C 17 01   STY $0117   und Adresse 1 setzen
```

```
A4BF    8D 18 01   STA $0118
A4C2    A9 02      LDA #$02      Status für Adresse 1 gelesen setzen
A4C4    05 81      ORA $81
A4C6    85 81      STA $81
A4C8    D0 31      BNE $A4FB     Unbedingter Sprung

A4CA    A9 04      LDA #$04      Adresse 2 schon gelesen ?
A4CC    20 22 A6   JSR $A622     Wenn ja, Fehler
A4CF    20 05 A6   JSR $A605     Adreßwert holen
A4D2    8C 19 01   STY $0119     und Adresse 2 setzen
A4D5    8D 1A 01   STA $011A
A4D8    A9 04      LDA #$04      Status für Adresse 2 gelesen
A4DA    D0 E8      BNE $A4C4     Status setzen

A4DC    A9 01      LDA #$01      Status für Filename 1 schon gelesen
A4DE    20 B9 A5   JSR $A5B9     Status testen und String holen
A4E1    8D 11 01   STA $0111     Filenamenlänge setzen
A4E4    A0 00      LDY #$00      Offset auf 0 setzen
A4E6    20 B7 03   JSR $03B7     Zeichen aus String holen
A4E9    8D 03 FF   STA $FF03     Write in Bank 0 setzen
A4EC    99 B7 12   STA $12B7,Y   Filename byteweise übertragen
A4EF    C8         INY           Offset erhöhen
A4F0    CC 11 01   CPY $0111     Filename kopiert ?
A4F3    90 F1      BCC $A4E6     Nein, nochmal
A4F5    A9 01      LDA #$01      Status für Filname 1 gelesen setzen
A4F7    05 80      ORA $80
A4F9    85 80      STA $80

A4FB    20 86 03   JSR $0386     CHRGOT
A4FE    D0 19      BNE $A519     Wenn kein Trennzeichen, Skip
A500    4C EA A3   JMP $A3EA     Ende ausführen

A503    C9 91      CMP #$91      Token für ON ?
A505    D0 03      BNE $A50A     Nein, Skip
A507    4C 37 A4   JMP $A437     Unitnummer einlesen
A50A    C9 A4      CMP #$A4      Token für TO
A50C    F0 02      BEQ $A510     Ja, Skip
A50E    D0 6D      BNE $A57D     Nein, Fehler

A510    20 80 03   JSR $0380     CHRGOT
A513    C9 50      CMP #$50      'P' Adreßangabe ?
A515    D0 0F      BNE $A526     Nein, Skip
A517    F0 B1      BEQ $A4CA     Ja, Skip

A519    C9 2C      CMP #$2C      ',' ?
A51B    D0 E6      BNE $A503     Nein, Test auf andere Zeichen
A51D    20 80 03   JSR $0380     Zeichen überlesen
```

```
A520   4C F8 A3   JMP $A3F8    und weiter testen

A523   20 80 03   JSR $0380    CHRGET
A526   C9 44      CMP #$44     'D' Drivenummer 2 ?
A528   F0 10      BEQ $A53A    Ja, Skip
A52A   C9 91      CMP #$91     Token für ON ?
A52C   F0 1F      BEQ $A54D    Ja, Skip
A52E   C9 55      CMP #$55     'U' Unitnummr 2 ?
A530   F0 21      BEQ $A553    Ja, Skip
A532   C9 22      CMP #$22     '"' Filename 2 ?
A534   F0 22      BEQ $A558    Ja, Skip
A536   C9 28      CMP #$28     '(' Filenamenausdruck 2 ?
A538   F0 1E      BEQ $A558    Ja, SKip

A53A   A9 20      LDA #$20     Laufwerksnummer 2 schon gelesen ?
A53C   20 1D A6   JSR $A61D    Wenn ja, Fehler
A53F   20 F2 A5   JSR $A5F2    Laufwerksnummer holen
A542   E0 02      CPX #$02     Zu groß ?
A544   B0 39      BCS $A57F    Ja, Fehler
A546   8E 14 01   STX $0114    Laufwerksnummer 2 setzen
A549   A9 20      LDA #$20     Status für Laufwerksnummer 2 gelesen
A54B   D0 1B      BNE $A568    Status setzen

A54D   20 82 A5   JSR $A582    Routine für ON aufrufen
A550   4C 68 A5   JMP $A568    Und weiter testen

A553   20 8D A5   JSR $A58D    Routine für 'U' aufrufen
A556   D0 10      BNE $A568    Und weiter testen

A558   A9 02      LDA #$02     Status für Filename 2 gelesen
A55A   20 B9 A5   JSR $A5B9    Status testen und Filename 2 lesen
A55D   8D 13 01   STA $0113    Länge setzen
A560   8E 15 01   STX $0115    Adresse setzen
A563   8C 16 01   STY $0116
A566   A9 02      LDA #$02     Status für Filename 2 gelesen laden

A568   05 80      ORA $80      Status setzen
A56A   85 80      STA $80
A56C   20 86 03   JSR $0386    CHRGET
A56F   F0 8F      BEQ $A500    Wenn Trennzeichen, Ende
A571   C9 2C      CMP #$2C     ',' ?
A573   F0 AE      BEQ $A523    Ja, weitermachen
A575   C9 91      CMP #$91     Token für ON ?
A577   F0 D4      BEQ $A54D    Ja, Routine für ON aufrufen
A579   C9 55      CMP #$55     'U' ?
A57B   F0 D6      BEQ $A553    Ja, Routine für 'U' aufrufen
A57D   D0 37      BNE $A5B6    Sonst Fehler
```

```
A57F   4C 28 7D   JMP $7D28     'ILLEGAL QUANTITY'

****************************** Routine für Token ON

A582   20 80 03   JSR $0380     CHRGET
A585   C9 42      CMP #$42      'B' ?
A587   F0 15      BEQ $A59E     Ja, Skip
A589   C9 55      CMP #$55      'U' ?
A58B   D0 29      BNE $A5B6     Nein, Fehler

****************************** Routine für 'U' Unitnummer holen

A58D   20 F2 A5   JSR $A5F2     Unitnummer holen
A590   E0 1F      CPX #$1F      Wert zu groß ?
A592   B0 56      BCS $A5EA     Ja, Fehler
A594   E0 04      CPX #$04      Wert zu klein ?
A596   90 52      BCC $A5EA     Ja, Fehler
A598   8E 1C 01   STX $011C     Gerätenummer setzen
A59B   A9 08      LDA #$08      Status für Gerätenummer gelesen, setzen
A59D   60         RTS

****************************** Routine für 'B' Bank holen

A59E   A9 01      LDA #$01      Bank-Wert schon gelesen ?
A5A0   20 22 A6   JSR $A622     Wenn ja, Fehler
A5A3   20 F2 A5   JSR $A5F2     Bank-Wert holen
A5A6   E0 10      CPX #$10      Bank zu groß ?
A5A8   B0 D5      BCS $A57F     Ja, Fehler
A5AA   8E 1F 01   STX $011F     Bank-Wert setzen
A5AD   A9 01      LDA #$01      Status für Bank-Wert gelesen setzen
A5AF   05 81      ORA $81
A5B1   85 81      STA $81
A5B3   A9 00      LDA #$00
A5B5   60         RTS
A5B6   4C 6C 79   JMP $796C     'SYNTAX'

****************************** Status testen und Filenamen lesen

A5B9   20 1D A6   JSR $A61D     Status testen
A5BC   20 7B 87   JSR $877B     FRMEVL + FRESTR Stringausdruck holen
A5BF   AA         TAX           Länge = 0 ?
A5C0   F0 25      BEQ $A5E7     Ja, Fehler
A5C2   A0 00      LDY #$00      Offset auf 0
A5C4   20 B7 03   JSR $03B7     Erstes Zeichen aus String lesen
A5C7   C9 40      CMP #$40      Klammeraffe ?
A5C9   D0 12      BNE $A5DD     Nein, Skip
A5CB   A9 80      LDA #$80      Klammeraffe schon gelesen ?
A5CD   20 1D A6   JSR $A61D     Wenn ja, Fehler
```

```
A5D0   A5 80      LDA $80      Status für Klammeraffe setzen
A5D2   09 80      ORA #$80
A5D4   85 80      STA $80
A5D6   CA         DEX          Länge erniedrigen
A5D7   E6 24      INC $24      und Startadresse des Strings erhöhen
A5D9   D0 02      BNE $A5DD    Kein Übertrag, Skip
A5DB   E6 25      INC $25      Übertrag berücksichtigen
A5DD   8A         TXA          Länge laden
A5DE   C9 11      CMP #$11     Länge > 16 ?
A5E0   B0 0B      BCS $A5ED    Ja, Fehler
A5E2   A6 24      LDX $24      Startadresse laden
A5E4   A4 25      LDY $25
A5E6   60         RTS

******************************* Verschiedene Fehlermeldungen

A5E7   A2 08      LDX #$08     'MISSING FILENAME'
A5E9   2C         .BYTE $2C    Nächsten Befehl überlesen
A5EA   A2 09      LDX #$09     'ILLEGAL DEVICE NUMBER'
A5EC   2C         .BYTE $2C    Nächsten Befehl überlesen
A5ED   A2 17      LDX #$17     'STRING TOO LONG'
A5EF   4C 3C 4D   JMP $4D3C    Fehler ausgeben

******************************* Byte-Wert in (X) holen

A5F2   20 80 03   JSR $0380    CHRGET
A5F5   F0 BF      BEQ $A5B6    Wenn Trennzeichen, Fehler
A5F7   90 09      BCC $A602    Wenn Ziffer, Skip
A5F9   20 59 79   JSR $7959    Test auf '('
A5FC   20 F4 87   JSR $87F4    Byte-Wert holen
A5FF   4C 56 79   JMP $7956    Test auf ')'
A602   4C F4 87   JMP $87F4    Byte-Wert holen

******************************* Ausdruck in (Y)/(A) holen

A605   20 80 03   JSR $0380    CHRGOT
A608   F0 AC      BEQ $A5B6    Wenn Trennzeichen, Fehler
A60A   90 0E      BCC $A61A    Wenn Ziffer, Skip
A60C   20 59 79   JSR $7959    Test auf '('
A60F   20 12 88   JSR $8812    Ausdruck holen
A612   20 56 79   JSR $7956    Test auf ')'
A615   A4 16      LDY $16      Ausdruck laden
A617   A5 17      LDA $17
A619   60         RTS
A61A   4C 12 88   JMP $8812    Ausdruck holen
```

```
******************************** Parameterstatus testen

A61D   25 80      AND $80      Status schon gesetzt ?
A61F   D0 95      BNE $A5B6    Ja, Fehler
A621   60         RTS

******************************** Adreßstatus testen

A622   25 81      AND $81      Status schon gesetzt ?
A624   D0 90      BNE $A5B6    Ja, Fehler
A626   60         RTS

******************************** Stringtabelle für Diskbefehle

A627   49 D1                   'I' (Drive 1)

A629   24 D1 3A F1             '$' (Drive 1) ':' (Filename 1)

A62D   F0 D1 3A F1 2C E1 2C E0 (Klammeraffe) (Drive 1) ':' (Filename 1)
                               ',' (Dateiart) (Dateityp)
A635   43 D2 3A F2 3D D2 3A F2 'C' (Drive 2) ':' (Filename 2)
                               '=' (Drive 2) ':' (Filename 2)

A63D   2C                      ','

A63E   D1 3A F1 2C 41          (Drive 1) ':' (Filename 1) ',A'

A643   4E D1 3A F1 2C D0       'N' (Drive 1) ':' (Filename 1) ',' (ID)

A649   56 D1                   'V' (Drive 1)

A64B   44 D2 3D D1             'D' (Drive 2) '=' (Drive 1)

A64F   43 D2 3A F2 3D D1 3A F1 'C' (Drive 2) ':' (Filename 2) '='
                               (Drive 1) ':' (Filename 1)

A657   52 D1 3A F2 3D D1 3A F1 'R' (Drive 1) ':' (Filename 2) '='
                               (Drive 1) ':' (Filename 1)

A65F   53 D1 3A F1             'S' (Drive 1) ':' (Filename 1)

A663   50 C2 E2 E0             'P' (Sekundäradresse) (Satznummer) (Byte)

******************************** Diskstring zusammenstellen

A667   8D 10 01   STA $0110    Länge setzen
A66A   98         TYA          Offset retten
A66B   48         PHA
```

```
A66C   20 0D A8   JSR $A80D     DS$ für ungültig erklären
A66F   A2 00      LDX #$00
A671   68         PLA           Offset weider holen
A672   CE 10 01   DEC $0110     Ende erreicht ?
A675   30 48      BMI $A6BF     Ja, Skip
A677   A8         TAY           Offset setzen
A678   C8         INY           Offset erhöhen
A679   98         TYA           Und wieder retten
A67A   48         PHA
A67B   B9 27 A6   LDA $A627,Y   Zeichen aus Tabelle holen
A67E   10 37      BPL $A6B7     Wenn normales Zeichen, eintragen
A680   C9 C2      CMP #$C2      Sekundäradresse ?
A682   F0 52      BEQ $A6D6     Ja, Skip
A684   C9 D0      CMP #$D0      Code für Disk ID ?
A686   F0 5D      BEQ $A6E5     Ja, Skip
A688   C9 E2      CMP #$E2      Satznummer ?
A68A   F0 77      BEQ $A703     Ja, Skip
A68C   C9 E1      CMP #$E1      Code für Dateiart ?
A68E   F0 61      BEQ $A6F1     Ja, Skip
A690   C9 F0      CMP #$F0      Code für Klammeraffe ?
A692   F0 47      BEQ $A6DB     Ja, Skip
A694   C9 F1      CMP #$F1      Code für Filename 1 ?
A696   F0 75      BEQ $A70D     Ja, Skip
A698   C9 F2      CMP #$F2      Code für Filename 2 ?
A69A   F0 21      BEQ $A6BD     Ja, Skip
A69C   C9 E0      CMP #$E0      Code für Dateityp ?
A69E   D0 05      BNE $A6A5     Nein, Skip
A6A0   AD 1E 01   LDA $011E     Dateityp/Satzlänge laden
A6A3   D0 12      BNE $A6B7     Unbedingter Sprung

A6A5   C9 D1      CMP #$D1      Code für Drivenummer 1 ?
A6A7   D0 05      BNE $A6AE     Nein, Skip
A6A9   AD 12 01   LDA $0112     Drivenummer 1 laden
A6AC   10 07      BPL $A6B5     Unbedingter Sprung

A6AE   C9 D2      CMP #$D2      Code für Drivenummer 2 ?
A6B0   D0 BF      BNE $A671     Nein, Zeichen überlesen
A6B2   AD 14 01   LDA $0114     Drivenummer 2 laden
A6B5   09 30      ORA #$30      und in ASCII wandeln

A6B7   9D 00 11   STA $1100,X   Zeichen eintragen
A6BA   E8         INX           Zieloffset erhöhen
A6BB   D0 B4      BNE $A671     Unbedingter Sprung

A6BD   F0 64      BEQ $A723     Filenamen 2 einlesen

A6BF   8A         TXA           Länge des Ergebnisstrings
A6C0   48         PHA           retten
```

```
A6C1   A2 00      LDX #$00      Adresse auf $1100
A6C3   A0 11      LDY #$11      setzen
A6C5   20 5D 92   JSR $925D     Filenamenparameter setzen
A6C8   AD 1B 01   LDA $011B     Filenummer laden
A6CB   AE 1C 01   LDX $011C     Gerätenummer laden
A6CE   AC 1D 01   LDY $011D     Sekundäradresse laden
A6D1   20 57 92   JSR $9257     Fileparameter setzen
A6D4   68         PLA           Länge wieder holen
A6D5   60         RTS

A6D6   AD ED 11   LDA $11ED     Sekundäradresse laden
A6D9   D0 DC      BNE $A6B7     Unbedingter Sprung

A6DB   24 80      BIT $80       Klammeraffe vorhanden ?
A6DD   30 02      BMI $A6E1     Ja, Skip
A6DF   10 90      BPL $A671     Nein, weitermachen

A6E1   A9 40      LDA #$40      Klammeraffe eintragen
A6E3   D0 D2      BNE $A6B7     Unbedingter Sprung

A6E5   AD 20 01   LDA $0120     Erstes Zeichen der ID laden
A6E8   9D 00 11   STA $1100,X   und eintragen
A6EB   E8         INX           Offset erhöhen
A6EC   AD 21 01   LDA $0121     und zweites Zeichen eintragen
A6EF   D0 C6      BNE $A6B7     Unbedingter Sprung

A6F1   AD 1E 01   LDA $011E     Satzlänge angegeben ?
A6F4   F0 04      BEQ $A6FA     Nein, Skip
A6F6   A9 4C      LDA #$4C      'L' Satzlänge für REL-Datei
A6F8   D0 BD      BNE $A6B7     Unbedingter Sprung
A6FA   A9 53      LDA #$53      'S' Dateityp sequentiell laden
A6FC   8D 1E 01   STA $011E     und setzen
A6FF   A9 57      LDA #$57      'W'
A701   D0 B4      BNE $A6B7     Write eintragen

A703   A5 16      LDA $16       Satzlänge eintragen
A705   9D 00 11   STA $1100,X   Zeichen eintragen
A708   A5 17      LDA $17       Zweites Byte
A70A   E8         INX           Offset erhöhen
A70B   D0 AA      BNE $A6B7     Unbedingter Sprung

A70D   AC 11 01   LDY $0111     Länge Filename 1 = 0 ?
A710   F0 33      BEQ $A745     Ja, Skip
A712   A0 00      LDY #$00      Filenamen eintragen
A714   B9 B7 12   LDA $12B7,Y   Filename laden
A717   9D 00 11   STA $1100,X   und kopieren
A71A   E8         INX           Offset erhöhen
A71B   C8         INY           Offset erhöhen
```

```
A71C   CC 11 01   CPY $0111     Länge erreicht ?
A71F   D0 F3      BNE $A714     Nein, weiter kopieren
A721   F0 23      BEQ $A746     Zurück zur Kopierschleife

A723   AD 15 01   LDA $0115     Adresse des zweiten Filenamens setzen
A726   85 24      STA $24
A728   AD 16 01   LDA $0116
A72B   85 25      STA $25
A72D   AC 13 01   LDY $0113     Länge = 0 ?
A730   F0 13      BEQ $A745     Ja, Skip
A732   A0 00      LDY #$00      Offset auf 0 setzen
A734   20 B7 03   JSR $03B7     Filenamen byteweise kopieren
A737   8D 03 FF   STA $FF03     Write in Bank 0 setzen
A73A   9D 00 11   STA $1100,X   Filename setzen
A73D   E8         INX           Offset erhöhen
A73E   C8         INY           Offset erhöhen
A73F   CC 13 01   CPY $0113     Länge erreicht ?
A742   D0 F0      BNE $A734     Nein, weiter kopieren
A744   24         .BYTE $24     Nächsten Befehl überlesen
A745   CA         DEX           Ein Zeichen zurückschalten
A746   4C 71 A6   JMP $A671     und zum Anfang der Schleife
```

****************************** Syntaxprüfung auf Name 1, Drive 1 und Unit

```
A749   29 E6      AND #$E6      Falsche Parameter angegeben ?
A74B   F0 03      BEQ $A750     Nein, Skip
A74D   4C 6C 79   JMP $796C     'SYNTAX'
A750   A5 80      LDA $80       Flag für Parameter laden
A752   29 01      AND #$01      Dateiname 1 vorhanden ?
A754   C9 01      CMP #$01
A756   D0 F5      BNE $A74D     Nein, Fehler
A758   A5 80      LDA $80       Flag für Parameter laden
A75A   60         RTS
```

****************************** Prüfung auf anderes als Name 1 und Drive 1

```
A75B   29 E7      AND #$E7      Falsche Parameter angegeben ?
A75D   D0 EE      BNE $A74D     Ja, Fehler
A75F   60         RTS
```

****************************** Prüfung auf beide Dateinamen und Drives

```
A760   29 C4      AND #$C4      Falsche Parameter angegeben ?
A762   D0 E9      BNE $A74D     Ja, Fehler
A764   A5 80      LDA $80       Flag für Parameter laden
A766   29 03      AND #$03
```

```
A768   C9 03      CMP #$03      Beide Dateinamen angegeben ?
A76A   D0 E1      BNE $A74D     Nein, Fehler
A76C   A5 80      LDA $80       Flag für Parameter laden
A76E   60         RTS

****************************** Prüfung auf Name 1 und Filenummer

A76F   29 05      AND #$05
A771   C9 05      CMP #$05      Name 1 und Filenummer angegeben ?
A773   D0 D8      BNE $A74D     Nein, Fehler
A775   A5 80      LDA $80       Flag für Parameter laden
A777   60         RTS

****************************** DS$ holen

A778   A5 7A      LDA $7A       DS$ gesetzt ?
A77A   D0 19      BNE $A795     Ja, Skip
A77C   A9 28      LDA #$28      Deskriptor auf 40 Zeichen setzen
A77D   85 7A      STA $7A
A780   20 99 92   JSR $9299     Platz für String reservieren
A783   86 7B      STX $7B       Und Adresse setzen
A785   84 7C      STY $7C
A787   A0 28      LDY #$28      Pointer auf Trailer setzen
A789   8D 04 FF   STA $FF04     Write in Bank 1 setzen
A78C   A9 7A      LDA #$7A
A78E   91 7B      STA ($7B),Y   Trailer auf $007A setzen
A790   C8         INY           Offset erhöhen
A791   A9 00      LDA #$00      Und damit
A793   91 7B      STA ($7B),Y   String für gültig erklären
A795   AE 1C 01   LDX $011C     Geräteadresse gesetzt ?
A798   D0 05      BNE $A79F     Ja, Skip
A79A   A2 08      LDX #$08      Geräteadresse auf 8
A79C   8E 1C 01   STX $011C     setzen
A79F   A9 00      LDA #$00      Filenummer 0
A7A1   A0 6F      LDY #$6F      Sekundäradresse 15
A7A3   20 57 92   JSR $9257     Fileparameter setzen
A7A6   A9 00      LDA #$00      Filenamenlänge = 0 setzen
A7A8   20 5D 92   JSR $925D     Filenamenparameter setzen
A7AB   20 D8 90   JSR $90D8     OPEN
A7AE   A2 00      LDX #$00      Filenummer 0
A7B0   20 C6 FF   JSR $FFC6     Eingabegerät setzen
A7B3   B0 20      BCS $A7D5     Wenn Fehler, Skip
A7B5   A0 FF      LDY #$FF      Offset auf Start
A7B7   C8         INY           Offset erhöhen
A7B8   20 63 92   JSR $9263     BASIN
A7BB   8D 04 FF   STA $FF04     Write in Bank 1 setzen
A7BE   C9 0D      CMP #$0D      CR ?
A7C0   F0 06      BEQ $A7C8     Ja, Ende
```

```
A7C2   91 7B       STA ($7B),Y  Zeichen in DS$ speichern
A7C4   C0 28       CPY #$28     Länge schon = 40 ?
A7C6   90 EF       BCC $A7B7    Nein, weiter einlesen
A7C8   A9 00       LDA #$00     Null-Byte
A7CA   91 7B       STA ($7B),Y  als Abschluß in DS$
A7CC   20 6F 92    JSR $926F    CLRCH I/O rücksetzen
A7CF   A9 00       LDA #$00     Filenummer 0
A7D1   38          SEC
A7D2   4C 75 92    JMP $9275    CLOSE

A7D5   48          PHA          Fehler retten
A7D6   20 C8 A7    JSR $A7C8    DS$ beenden, CLOSE
A7D9   20 0D A8    JSR $A80D    DS$ für ungültig erklären
A7DC   68          PLA          Fehler laden
A7DD   AA          TAX          und für Ausgabe in (X)
A7DE   4C 3C 4D    JMP $4D3C    Fehler ausgeben

******************************* Im Direktmodus 'ARE YOU SURE'

A7E1   24 7F       BIT $7F      Direktmodus ?
A7E3   30 25       BMI $A80A    Nein, Skip
A7E5   20 81 92    JSR $9281    Text ausgeben

******************************* Textkonstante

A7E8   41 52 45 20 59 4F 55 20 'ARE YOU SURE?'
A7F0   53 55 52 45 3F 00

*******************************

A7F6   20 6F 92    JSR $926F    CLRCH I/O rücksetzen
A7F9   20 63 92    JSR $9263    BASIN Zeichen holen
A7FC   48          PHA          und retten
A7FD   C9 0D       CMP #$0D     CR ?
A7FF   F0 05       BEQ $A806    Ja, Skip
A801   20 63 92    JSR $9263    Zeichen überlesen
A804   D0 F7       BNE $A7FD    Bis Ende gefunden
A806   68          PLA          Erstes Zeichen wieder holen
A807   C9 59       CMP #$59     'Y' ? Flags setzen
A809   60          RTS

*******************************

A80A   A9 00       LDA #$00     (A) löschen
A80C   60          RTS
```

```
******************************** DS$ für ungültig erklären

A80D   98         TYA          (Y) retten
A80E   48         PHA
A80F   A5 7A      LDA $7A      Deskriptor gesetzt ?
A811   F0 0D      BEQ $A820    Nein, Skip
A813   A0 28      LDY #$28     (Y) = Länge von DS$ (40 Zeichen)
A815   98         TYA          Länge in (A)
A816   8D 04 FF   STA $FF04    Write in Bank 1 setzen
A819   91 7B      STA ($7B),Y  Trailer setzen
A81B   C8         INY          Offset erhöhen
A81C   A9 FF      LDA #$FF     String für ungültig erklären
A81E   91 7B      STA ($7B),Y  Trailerbyte setzen
A820   A9 00      LDA #$00
A822   8D 03 FF   STA $FF03    Write in Bank 0 setzen
A825   85 7A      STA $7A      Deskriptor löschen
A827   68         PLA
A828   A8         TAY          (Y) wieder holen
A829   60         RTS

******************************** Textkonstante 'KEY 0,'

A82A   2C 30 20 59 45 4B       ',0 YEK' ergibt 'KEY 0,'

******************************** Byte-Wert in (A) ausgeben

A830   AA         TAX          Byte-Wert in (X)
A831   98         TYA          (Y) retten
A832   48         PHA
A833   A9 00      LDA #$00     High-Byte auf 0 setzen
A835   20 32 8E   JSR $8E32    Adresse in (X)/(A) ausgeben
A838   68         PLA
A839   A8         TAY          (Y) wieder holen
A83A   60         RTS

******************************** Direktmodusflags setzen falls nötig

A83B   85 3C      STA $3C      Zeilennummer High-Byte setzen
A83D   88         DEY
A83E   AA         TAX
A83F   E8         INX          War High-Byte = $FF ? (Direktmodus)
A840   D0 02      BNE $A844    Nein, Ende
A842   86 7F      STX $7F      Direktmodusflag
A844   60         RTS

******************************** ROMs einschalten

A845   48         PHA          (A) retten
```

```
A846   A9 00      LDA #$00
A848   8D 00 FF   STA $FF00    Alle ROMs einschalten
A84B   68         PLA          (A) wieder holen
A84C   60         RTS

******************************* BASIC-IRQ Routine

A84D   AD FD 12   LDA $12FD    BASIC-IRQ erlaubt ?
A850   F0 01      BEQ $A853    Ja, Skip
A852   60         RTS
A853   EE FD 12   INC $12FD    IRQ verbieten
A856   A2 10      LDX #$10     Spriteadressen in VIC kopieren
A858   BD D6 11   LDA $11D6,X  Bytes aus Tabelle laden
A85B   9D 00 D0   STA $D000,X  und VIC programmieren
A85E   CA         DEX          Alle Bytes kopiert ?
A85F   10 F7      BPL $A858    Nein,nochmal
A861   A0 07      LDY #$07     8 Sprites bearbeiten
A863   AD 15 D0   LDA $D015    VIC-Spriteschalter laden
A866   39 B3 6C   AND $6CB3,Y  Sprite angeschaltet ?
A869   F0 38      BEQ $A8A3    Nein, Skip
A86B   BE D9 6D   LDX $6DD9,Y  Tabellenoffset auf Spritedaten laden
A86E   BD 7E 11   LDA $117E,X  Soll Sprite bewegt werden ?
A871   F0 30      BEQ $A8A3    Nein, Skip
A873   9D 7F 11   STA $117F,X  Geschwindigkeitszähler setzen
A876   98         TYA
A877   0A         ASL
A878   A8         TAY
A879   BD 80 11   LDA $1180,X
A87C   38         SEC
A87D   E9 01      SBC #$01
A87F   E8         INX
A880   E8         INX
A881   C8         INY
A882   20 F4 A9   JSR $A9F4    Position aufaddieren
A885   CA         DEX
A886   CA         DEX
A887   88         DEY
A888   BD 80 11   LDA $1180,X
A88B   20 F4 A9   JSR $A9F4    Position aufaddieren
A88E   08         PHP          Übertrag retten
A88F   98         TYA
A890   4A         LSR
A891   A8         TAY
A892   28         PLP          Übertrag vorhanden ?
A893   90 09      BCC $A89E    Nein, Skip
A895   AD E6 11   LDA $11E6    MSB der Spriteposition setzen
A898   59 B3 6C   EOR $6CB3,Y
A89B   8D E6 11   STA $11E6
```

```
A89E   DE 7F 11   DEC $117F,X  Zähler = 0 ?
A8A1   D0 D3      BNE $A876    Nein, weiterbewegen
A8A3   88         DEY          Alle Sprites verarbeitet ?
A8A4   10 BD      BPL $A863    Nein, weiter testen
A8A6   AD 19 D0   LDA $D019    VIC-Interrupts löschen
A8A9   8D 19 D0   STA $D019    VIC programmieren
A8AC   29 0E      AND #$0E     War etwas gesetzt ?
A8AE   F0 44      BEQ $A8F4    Nein, Skip
A8B0   4A         LSR          Wert anpassen
A8B1   A0 01      LDY #$01     Nummer laden
A8B3   4A         LSR          Spritekollision ?
A8B4   90 20      BCC $A8D6    Nein, Skip
A8B6   48         PHA          IRQ-Wert retten
A8B7   B9 1E D0   LDA $D01E,Y  Kollisionsbits
A8BA   19 E7 11   ORA $11E7,Y  In Interpreterwerte setzen
A8BD   99 E7 11   STA $11E7,Y
A8C0   A9 00      LDA #$00
A8C2   99 1E D0   STA $D01E,Y  Kollisionsbits löschen
A8C5   AD 7F 12   LDA $127F    BASIC-IRQ Flag laden
A8C8   C0 00      CPY #$00     Sprite/Sprite Kollision ?
A8CA   F0 01      BEQ $A8CD    Ja, Skip
A8CC   4A         LSR
A8CD   4A         LSR          Art der Kollision erlaubt ?
A8CE   90 05      BCC $A8D5    Nein, Skip
A8D0   A9 FF      LDA #$FF     BASIC-IRQ Wert setzen
A8D2   99 76 12   STA $1276,Y
A8D5   68         PLA
A8D6   88         DEY          Beide Kollisionsarten getestet ?
A8D7   10 DA      BPL $A8B3    Nein, weitermachen
A8D9   4A         LSR          Lightpenaktivierung ?
A8DA   90 18      BCC $A8F4    Nein, Skip
A8DC   AD 13 D0   LDA $D013    Lightpenposition retten
A8DF   8D E9 11   STA $11E9
A8E2   AD 14 D0   LDA $D014
A8E5   8D EA 11   STA $11EA
A8E8   AD 7F 12   LDA $127F
A8EB   29 04      AND #$04     Lightpen-IRQ erlaubt ?
A8ED   F0 05      BEQ $A8F4    Nein, Skip
A8EF   A9 FF      LDA #$FF     Lightpenflag setzen
A8F1   8D 78 12   STA $1278

A8F4   A2 00      LDX #$00
A8F6   BD 24 12   LDA $1224,X  Stimme aktiv ?
A8F9   30 27      BMI $A922    Nein, Skip
A8FB   BD 23 12   LDA $1223,X  Von Stimmenzähler Tempo abziehen
A8FE   38         SEC
A8FF   ED 22 12   SBC $1222
A902   9D 23 12   STA $1223,X
```

```
A905  B0 1B     BCS $A922    Wenn kein Übertrag, Skip
A907  BD 24 12  LDA $1224,X
A90A  E9 00     SBC #$00
A90C  9D 24 12  STA $1224,X
A90F  B0 11     BCS $A922    Wenn kein Übertrag, Skip
A911  8A        TXA
A912  4A        LSR
A913  A8        TAY
A914  B9 30 12  LDA $1230,Y  Wellenform laden
A917  29 FE     AND #$FE     KEY-Bit löschen
A919  48        PHA
A91A  B9 39 70  LDA $7039,Y  Stimmenoffset setzen
A91D  A8        TAY
A91E  68        PLA
A91F  99 04 D4  STA $D404,Y  Und Stimme ausklingen lassen
A922  E8        INX          Offset erhöhen
A923  E8        INX          Offset erhöhen
A924  E0 06     CPX #$06     Alle 3 Stimmen verarbeitet ?
A926  D0 CE     BNE $A8F6    Nein, weiter testen
A928  A0 02     LDY #$02     Offset laden
A92A  B9 85 12  LDA $1285,Y  Soundstimme gesetzt ?
A92D  10 06     BPL $A935    Ja, Skip
A92F  88        DEY          Alle 3 Soundstimmen getestet ?
A930  10 F8     BPL $A92A    Nein, weiter
A932  4C F0 A9  JMP $A9F0    BASIC-IRQ wieder erlauben und Ende
A935  18        CLC
A936  B9 9D 12  LDA $129D,Y  Schrittzahl auf Frequenz aufaddieren
A939  79 97 12  ADC $1297,Y
A93C  99 9D 12  STA $129D,Y
A93F  B9 A0 12  LDA $12A0,Y
A942  79 9A 12  ADC $129A,Y
A945  99 A0 12  STA $12A0,Y
A948  B9 94 12  LDA $1294,Y  Richtung laden
A94B  AA        TAX
A94C  29 01     AND #$01     Zunehmend ?
A94E  F0 2E     BEQ $A97E    Ja, Skip
A950  90 0F     BCC $A961    Wenn kein alter Übertrag vorhanden, Skip
A952  38        SEC
A953  B9 9D 12  LDA $129D,Y  Minimalwert von Frequenz abziehen
A956  F9 8E 12  SBC $128E,Y
A959  B9 A0 12  LDA $12A0,Y
A95C  F9 91 12  SBC $1291,Y
A95F  B0 4D     BCS $A9AE    Wenn kein Übertrag, Skip
A961  E0 02     CPX #$02     Oszillierend ?
A963  90 0A     BCC $A96F    Nein, Skip
A965  20 DA A9  JSR $A9DA    Schrittzahl negieren
A968  A9 02     LDA #$02     Oszillierend zunehmend
A96A  99 94 12  STA $1294,Y  Setzen
```

```
A96D   D0 33      BNE $A9A2    Unbedingter Sprung

A96F   B9 88 12   LDA $1288,Y  Maximalwert in Frequenz setzen
A972   99 9D 12   STA $129D,Y
A975   B9 8B 12   LDA $128B,Y
A978   99 A0 12   STA $12A0,Y
A97B   4C AE A9   JMP $A9AE

A97E   B0 14      BCS $A994    Wenn alter Übertrag vorhanden, Skip
A980   B9 A0 12   LDA $12A0,Y  Maximalwert erreicht ?
A983   D9 8B 12   CMP $128B,Y
A986   90 26      BCC $A9AE    Nein, Skip
A988   D0 0A      BNE $A994
A98A   B9 9D 12   LDA $129D,Y
A98D   D9 88 12   CMP $1288,Y
A990   90 1C      BCC $A9AE    Nein, Skip
A992   F0 1A      BEQ $A9AE    Nein, Skip
A994   E0 02      CPX #$02     Oszillierend zunehmend ?
A996   90 0A      BCC $A9A2    Nein, Skip
A998   20 DA A9   JSR $A9DA    Schrittzahl negieren
A99B   A9 03      LDA #$03     Oszillierend abnehmend
A99D   99 94 12   STA $1294,Y  Setzen
A9A0   D0 CD      BNE $A96F    Maximalfrequenz setzen

A9A2   B9 8E 12   LDA $128E,Y  Minimalfrequenz setzen
A9A5   99 9D 12   STA $129D,Y
A9A8   B9 91 12   LDA $1291,Y
A9AB   99 A0 12   STA $12A0,Y

A9AE   BE 39 70   LDX $7039,Y  Stimmenoffset laden
A9B1   B9 9D 12   LDA $129D,Y  Frequenz programmieren
A9B4   9D 00 D4   STA $D400,X
A9B7   B9 A0 12   LDA $12A0,Y
A9BA   9D 01 D4   STA $D401,X
A9BD   98         TYA          (Y) in (X) kopieren
A9BE   AA         TAX
A9BF   BD 82 12   LDA $1282,X  Zeitspeicher erniedrigen
A9C2   D0 03      BNE $A9C7
A9C4   DE 85 12   DEC $1285,X
A9C7   DE 82 12   DEC $1282,X
A9CA   BD 85 12   LDA $1285,X  Noch Zeit vorhanden ?
A9CD   10 08      BPL $A9D7    Ja, Skip
A9CF   A9 08      LDA #$08     TEST-Bit
A9D1   BE 39 70   LDX $7039,Y  Offset auf SID-Stimme laden
A9D4   9D 04 D4   STA $D404,X  In SID setzen
A9D7   4C 2F A9   JMP $A92F    Nächste Soundstimme testen
```

```
******************************* Schrittzahl negieren

A9DA   B9 97 12   LDA $1297,Y  Low-Byte negieren
A9DD   49 FF      EOR #$FF
A9DF   18         CLC
A9E0   69 01      ADC #$01
A9E2   99 97 12   STA $1297,Y
A9E5   B9 9A 12   LDA $129A,Y  High-Byte negieren
A9E8   49 FF      EOR #$FF
A9EA   69 00      ADC #$00
A9EC   99 9A 12   STA $129A,Y
A9EF   60         RTS

******************************* BASIC-IRQ wieder erlauben

A9F0   CE FD 12   DEC $12FD    Flag auf 0 (war vorher 1)
A9F3   60         RTS

******************************* Spriteposition aufaddieren

A9F4   48         PHA
A9F5   18         CLC
A9F6   BD 81 11   LDA $1181,X
A9F9   7D 85 11   ADC $1185,X
A9FC   9D 85 11   STA $1185,X
A9FF   BD 82 11   LDA $1182,X
AA02   7D 86 11   ADC $1186,X
AA05   9D 86 11   STA $1186,X
AA08   68         PLA
AA09   90 13      BCC $AA1E    Wenn kein Übertrag, Skip
AA0B   4A         LSR
AA0C   4A         LSR
AA0D   B9 D6 11   LDA $11D6,Y  Position laden
AA10   B0 05      BCS $AA17    Wenn subtrahieren, Skip
AA12   69 01      ADC #$01     Position erhöhen
AA14   4C 1B AA   JMP $AA1B
AA17   E9 01      SBC #$01     Position erniedrigen
AA19   C9 FF      CMP #$FF     Auf Übertrag vergleichen
AA1B   99 D6 11   STA $11D6,Y  Und setzen
AA1E   60         RTS

******************************* BASIC-Befehl STASH

AA1F   A9 84      LDA #$84     Code für STASH
AA21   4C 2B AA   JMP $AA2B
```

```
****************************** BASIC-Befehl FETCH

AA24   A9 85      LDA #$85    Code für FETCH
AA26   4C 2B AA   JMP $AA2B

****************************** BASIC-Befehl SWAP

AA29   A9 86      LDA #$86    Code für SWAP
AA2B   48         PHA         Befehlscode retten
AA2C   20 12 88   JSR $8812   Anzahl der Bytes holen
AA2F   20 45 A8   JSR $A845   ROMs einschalten
AA32   8C 07 DF   STY $DF07   Anzahl an RAM-Disk übergeben
AA35   8D 08 DF   STA $DF08
AA38   20 0F 88   JSR $880F   Startadresse im Computer holen
AA3B   20 45 A8   JSR $A845   ROMs einschalten
AA3E   8C 02 DF   STY $DF02   Startadresse übergeben
AA41   8D 03 DF   STA $DF03
AA44   20 0F 88   JSR $880F   Startadresse in RAM-Disk holen
AA47   20 45 A8   JSR $A845   ROMs einschalten
AA4A   8C 04 DF   STY $DF04   Startadresse übergeben
AA4D   8D 05 DF   STA $DF05
AA50   20 09 88   JSR $8809   Banknummer in RAM-Disk holen
AA53   E0 10      CPX #$10    Banknummer zu groß ?
AA55   B0 0E      BCS $AA65   Ja, Fehler
AA57   20 45 A8   JSR $A845   ROMs einschalten
AA5A   8E 06 DF   STX $DF06   Bank programmieren
AA5D   68         PLA         Befehlscode wieder holen
AA5E   A8         TAY         und in (Y)
AA5F   AE D5 03   LDX $03D5   Bank des Computers laden
AA62   4C 50 FF   JMP $FF50   DMA starten
AA65   4C 28 7D   JMP $7D28   'ILLEGAL QUANTITY'

****************************** FAC#1 gerundet in Integerformat

AA68   20 47 8C   JSR $8C47   FAC#1 runden
AA6B   4C C7 8C   JMP $8CC7   und in Integer wandeln

****************************** Füllbytes

AA6E   FF FF FF FF . . .

****************************** Verschlüsselter Text

AE63   . . . . . . .  7B E9 77 Siehe FRE-Funktion ab $8000 !
AE66   6A 5F 5E 5D BE 21 3D 24 Sie sollten sich diesen Text einmal
AE6E   37 3F 22 55 20 24 4A 30 ausgeben lassen !
AE76   27 3A 4E 2F 35 4D 4C 4F
AE7E   40 47 46 68 69 88 15 1F
```

```
AE86    0C 08 1F 0F 19 69 5F 71
AE8E    96 05 13 11 74 89 05 1E
AE96    0D 01 43 6D 98 06 10 13
AE9E    19 67 94 1C 05 75 37 19
AEA6    EE 70 70 1D F9 61 66 66
AEAE    79 79 73 38 39 E3 6F 7B
AEB6    6C 78 6F 7F 69 19 2F 01
AEBE    E2 6E 6A 05 EC 5E 48 5D
AEC6    15 3F DA 5C 4A 56 32 D9
AECE    51 4E 58 5C 51 06 2A CF
AED6    5A 4E 40 46 2C D3 43 4D
AEDE    41 4E 47 08 09 E9 36 B0
AEE6    B6 B4 DE BC AE BE A1 DD
AEEE    B4 B8 B8 D2 A0 CB A7 A8
AEF6    A3 AA CE B9 A4 A6 AF CF
AEFE    ED E7

****************************** Sprungtabelle für diverse Routinen

AF00    4C B4 84    JMP $84B4    FAC#1 in Integer mit Bereichsprüfung
AF03    4C 3C 79    JMP $793C    Integer (A)/(Y) in FAC#1
AF06    4C 42 8E    JMP $8E42    FAC#1 in ASCII ab $0100
AF09    4C 52 80    JMP $8052    ASCII ab ($24) in FAC#1
AF0C    4C 15 88    JMP $8815    FAC#1 in Adreßformat
AF0F    4C 75 8C    JMP $8C75    Adreßwert in FAC#1
AF12    4C 2E 88    JMP $882E    FAC#1 = Konstante (A)/(Y) - FAC#1 Bank 1
AF15    4C 31 88    JMP $8831    FAC#1 = FAC#2 - FAC#1
AF18    4C 45 88    JMP $8845    FAC#1 = Konstante (A)/(Y) + FAC#1 Bank 1
AF1B    4C 48 88    JMP $8848    FAC#1 = FAC#2 + FAC#1
AF1E    4C 24 8A    JMP $8A24    FAC#1 = Konstante (A)/(Y) * FAC#1 Bank 1
AF21    4C 27 8A    JMP $8A27    FAC#1 = FAC#2 * FAC#1
AF24    4C 49 8B    JMP $8B49    FAC#1 = Konstante (A)/(Y) / FAC#1 Bank 1
AF27    4C 4C 8B    JMP $8B4C    FAC#1 = FAC#2 / FAC#1
AF2A    4C CA 89    JMP $89CA    LOG-Funktion
AF2D    4C FB 8C    JMP $8CFB    INT-Funktion
AF30    4C B7 8F    JMP $8FB7    SQR-Funktion
AF33    4C FA 8F    JMP $8FFA    Vorzeichenwechsel FAC#1
AF36    4C BE 8F    JMP $8FBE    FAC#1 = FAC#2 ^ Konstante (A)/(Y)
AF39    4C C1 8F    JMP $8FC1    FAC#1 = FAC#2 ^ FAC#1
AF3C    4C 33 90    JMP $9033    EXP-Funktion
AF3F    4C 09 94    JMP $9409    COS-Funktion
AF42    4C 10 94    JMP $9410    SIN-Funktion
AF45    4C 59 94    JMP $9459    TAN-Funktion
AF48    4C B3 94    JMP $94B3    ATN-Funktion
AF4B    4C 47 8C    JMP $8C47    FAC#1 runden
AF4E    4C 84 8C    JMP $8C84    ABS-Funktion
AF51    4C 57 8C    JMP $8C57    Vorzeichen von FAC#1 holen
AF54    4C 87 8C    JMP $8C87    Vergleich Konstante (A)/(Y) mit FAC#1
```

```
AF57    4C 37 84    JMP $8437    RND-Funktion
AF5A    4C B4 8A    JMP $8AB4    FAC#2 = Konstante (A)/(Y) Bank 1
AF5D    4C 89 8A    JMP $8A89    FAC#2 = Konstante (A)/(Y)
AF60    4C 85 7A    JMP $7A85    FAC#1 = Konstante (A)/(Y) Bank 1
AF63    4C D4 8B    JMP $8BD4    FAC#1 = Konstante (A)/(Y)
AF66    4C 00 8C    JMP $8C00    FACMEM (X)/(Y) mit Rundung
AF69    4C 28 8C    JMP $8C28    FAC#1 = FAC#2
AF6C    4C 38 8C    JMP $8C38    FAC#2 = FAC#1 mit Rundung
AF6F    4C 28 48    JMP $4828    ??? Tabelle der Hierarchiewerte
AF72    4C 30 9B    JMP $9B30    Linie von Start- zu Zielkoordinaten
ziehen
AF75    4C FB 9B    JMP $9BFB    Koordinaten setzen
AF78    4C 50 67    JMP $6750    Nächste Koordinate auf Kreisbogen
berechnen
AF7B    4C 9B 5A    JMP $5A9B    RUN-Befehl
AF7E    4C F3 51    JMP $51F3    PC auf Programmstart, CLR
AF81    4C F8 51    JMP $51F8    CLR-Befehl
AF84    4C D6 51    JMP $51D6    NEW-Befehl
AF87    4C 4F 4F    JMP $4F4F    Zeilenverkettung korrigieren
AF8A    4C 0A 43    JMP $430A    Umwandlung in Interpretercode
AF8D    4C 64 50    JMP $5064    Zeile suchen
AF90    4C F6 4A    JMP $4AF6    Interpreterschleifenstart
AF93    4C D7 78    JMP $78D7    Nächstes Element eines Ausdrucks holen
AF96    4C EF 77    JMP $77EF    FRMEVL Ausdruck auswerten
AF99    4C A6 5A    JMP $5AA6    RUN-Direkteinsprung
AF9C    4C 81 5A    JMP $5A81    Programm-Modus setzen
AF9F    4C A0 50    JMP $50A0    Zeilennummer lesen
AFA2    4C EA 92    JMP $92EA    Garbage-Collection
AFA5    4C CD 4D    JMP $4DCD    Direkteinsprung in Eingabeschleife

****************************** Füllbytes

AFA8    FF . . .
AFFF    . . . FF
```

9.12 Die Zeropage

In der Zeropage hat man zu PET's Zeiten alle Systemvariablen gespeichert. Dies sind beispielsweise die aktuelle Cursorposition, Angaben über das aktuelle Ausgabegerät usw. 256 Bytes reichten aus, um diese Informationen zu speichern, weshalb man diesem System-Bereich auch den Namen Zeropage verpaßte.

Heute sieht dies alles ganz anders aus. 256 Byte reichen schon lange nicht mehr aus, alle Systeminformationen zu speichern. Den Namen Zeropage hat man allerdings beibehalten, da er sich sehr gut eingeprägt hat.

Die Zeropage bietet sehr viele Möglichkeiten zur direkten Manipulation bzw. beinhaltet auch eine Flut von Informationen, auf die der Programmierer zurückgreifen kann (was er auch unbedingt tun sollte). Da diese Zeropage so immens wichtig ist, finden Sie auf den nun folgenden Seiten nähere Informationen zu den einzelnen Speicheradressen. Diese Informationen werden Ihnen sicherlich sehr hilfreich sein.

Allerdings werden einige Adressen in der Zeropage nur im Zusammenhang mit den entsprechenden Routinen im Kernal und dem BASIC-ROM-Listing deutlich. Deshalb ist es sehr wichtig, daß Sie sich die entsprechenden Passagen im ROM-Listing vor Manipulationen in der Zeropage näher betrachten.

Die Zeropage des Commodore 128 und die erweiterte Zeropage

0000:	0000	6510 Datenrichtung Prozessorport	
0001:	0001	6510 Datenregister Prozessorport	
0002:	0002	Speicher für Bank Byte	
0003:	0003	Speicher für Programm-Zähler Hi-Wert	
0004:	0004	Speicher für Programm-Zähler Lo-Wert	
0005:	0005	Speicher für CPU Status Register	
0006:	0006	Speicher für Akkumulator	
0007:	0007	Speicher für X-Register	
0008:	0008	Speicher für Y-Register	
0009:	0009	Speicher für den Stapelzeiger / Suchzeichen	
000A:	0010	Sucht Anführungszeichen am Ende eines Strings	
000B:	0011	Bildschirmspalte ab letztem TAB	
000C:	0012	Disketten-Flag: 0=LOAD, 1=VERIFY	
000D:	0013	Anzahl der Elemente, Eingabepufferzeiger	
000E:	0014	Standardvorgabe bei Felddimensionierungen (DIM)	
000F:	0015	Datentyp-Flag 1: $00=numerisch, $FF=String	
0010:	0016	Datentyp-Flag 2: $00=Integer, $80=Floatingpoint	
0011:	0017	Flag: LIST, DATAs lesen, Garbage Collection	
0012:	0018	Zeiger für FN Funktion, Variablentyp für FOR/NEXT	
0013:	0019	Input-Flag: $00=INPUT, $40=GET, $98=READ	
0014:	0020	Vorzeichen des ATN; Flag:Gleichheit bei Vergleich	
0015:	0021	Aktives I/O Gerät, Flag: INPUT Kommentar	
0016:	0022 - 0023	Zeilennummer, Ganzahliger Wert Lo/High	
0018:	0024	Zeiger auf temporären Stringstack	
0019:	0025 - 0026	Adresse des letzten temporären Strings	
001B:	0027 - 0029	3 Byte Stapel für kurzzeitige Strings	
001E:	0030 - 0032	3 Byte Stapel für kurzzeitige Strings	
0021:	0033 - 0035	3 Byte Stapel für kurzzeitige Strings	
0024:	0036 - 0037	2 Byte für Hilfszeiger Index 1	
0026:	0038 - 0039	2 Byte für Hilfszeiger Index 2	
0028:	0040 - 0044	Gleitpunktergebnis der Multiplikation	
002D:	0045 - 0046	Zeiger: Basic-Text Anfang	Lo/Hi
002F:	0047 - 0048	Zeiger: Anfang der Basic-Variablen	Lo/Hi
0031:	0049 - 0050	Zeiger: Anfang der Basic-Felder	Lo/Hi
0033:	0051 - 0052	Zeiger: Ende der Basic-Felder + 1	Lo/Hi
0035:	0053 - 0054	Zeiger: Anfang des Stringspeichers	Lo/Hi
0037:	0055 - 0056	Hilfszeiger bei der Stringspeicherung	Lo/Hi
0039:	0057 - 0058	Zeiger: Ende Stringspeicher, Var. Bank 1	Lo/Hi
003B:	0059 - 0060	Aktuelle Basic Zeilennummer	Lo/Hi
003D:	0061 - 0062	Zeiger auf Basic Text für CHRGET, CHRGOT	Lo/Hi
003F:	0063 - 0064	PRINT USING Hilfszeiger, Suchzeichen Zeiger	Lo/Hi
0041:	0065 - 0066	Aktuelle DATAs Zeilennummer	Lo/Hi
0043:	0067 - 0068	Zeiger auf die aktuelle DATA Adresse	Lo/Hi
0045:	0069 - 0070	Vektorzeiger für INPUT Routine	Lo/Hi
0047:	0071 - 0072	Aktueller Basic Variablenname	Lo/Hi

0049:	0073 - 0074	Zeiger auf Adresse der aktuellen Variablen Lo/Hi
004B:	0075 - 0076	Maske für AND, LIST Zeiger, FOR NEXT Zeiger
004D:	0077 - 0078	Zwischenspeicher für Programmzeiger Lo/Hi
004F:	0079	Maske für Vergleichsoperationen >:2, =:4, <:8
0050:	0080 - 0081	Variablenzeiger f.FN Definition, + für Garb. Coll.
0052:	0082 - 0084	Zeiger: Descriptor in Variablenliste b. Stringvgl.
0055:	0085	Help Flag:
0056:	0086 - 0087	Sprungvektor für Funktionsauswertungen
0058:	0088	Oldov
0059:	0089	Bereich für INSTRING Operationen / Hilfszeiger 1
005A:	0090 - 0091	Zeiger: Blockübertragung, DIM Initialisierung
005C:	0092 - 0093	Zeiger: Blockübertragung
005E:	0094	Hilfszeiger 2, Zeitweilig für Fließkomma Akku
005F:	0095 - 0096	Vorkomma-/Nachkommastellenzahl für Umwandlung
0061:	0097	Zeiger: Dezimalpunkt b. Einlesen von Ziffernketten
0062:	0098	Exponent Vorzeichen der gelesenen Zahl (neg. =$80)
0063:	0099	Gleitpunktakkumulator 1: Exponent
0064:	0100 - 0103	Gleitpunktakkumulator 1: Mantisse
0068:	0104	Gleitpunktakkumulator 1: Vorzeichen
0069:	0105	Zeiger: Polynomauswertung
006A:	0106	Gleitpunktakkumulator 2: Exponent
006B:	0107 - 0110	Gleitpunktakkumulator 2: Mantisse
006F:	0111	Gleitpunktakkumulator 2: Vorzeichen
0070:	0112	Ergebnisflag: Vorzeichenvergleich Akku 1 zu Akku 2
0071:	0113	Gleitpunktakkumulator 1: Kleinste Stelle (Rundung)
0072:	0114 - 0115	Zeiger: Kassettenpuffer
0074:	0116 - 0117	Offset Wert für AUTO Befehl, $00=ausgeschaltet
0076:	0118	Hires Flag: bei EIN Basicstart um 10k hochsetzen
0077:	0119	Spritenummer/Zähler für führende Nullen/Farbquelle
0078:	0120	Help Zähler
0079:	0121	Zwischenspeicher für indirektes Laden
007A:	0122 - 0124	Beschreibung der Fehlervariable DS$
007D:	0125 - 0126	Ende des Stacks während des Programmlaufs
007F:	0127	Modus Flag: $80=RUN Modus, $00=Direkt Modus
0080:	0128	USING Zeiger f. Dezimalpunkt,Statuswort DOS Parser
0081:	0129	Parstx
0082:	0130	Oldstx
0083:	0131	Aktuelle Farbe für Grafikmodus
0084:	0132	Multicolor Modus: Farbe 1
0085:	0133	Multicolor Modus: Farbe 2
0086:	0134	Vordergrundfarbe
0087:	0135 - 0136	Faktor des Abbildungsmaßstabes in X-Richtung
0089:	0137 - 0138	Faktor des Abbildungsmaßstabes in Y-Richtung
008B:	0139	Stop zeichnen, wenn nicht Hintergrundfarbe
008C:	0140 - 0141	Adresszeiger für Grafik Routinen
008E:	0142	Hilfsspeicher 1 für Grafik Routinen
008F:	0143	Hilfsspeicher 2 für Grafik Routinen
0090:	0144	Statuswort für Kernal Ein- Ausgabeoperationen

0090:	0144	Statuswort für Kernal Ein- Ausgabeoperationen
0091:	0145	Stop Flag: STOP Taste, RVS Taste
0092:	0146	Zeit Konstante für Kassetten Operationen
0093:	0147	Load Flag: $00=LOAD, $01=VERIFY
0094:	0148	serieller Bus Flag: Zeichen im Puffer
0095:	0149	Zeichen im Puffer für den seriellen Bus
0096:	0150	Sync Nr. für Kassette, EOT vom Band empfangen
0097:	0151	Temporäre Datenadresse
0098:	0152	Index für Dateitabellen, Zahl der offenen Dateien
0099:	0153	Standard Eingabegerät (0 für Tastatur)
009A:	0154	Standard Ausgabegerät (3 für Bildschirm)
009B:	0155	Paritätsbyte von Kassette
009C:	0156	Band Flag: Byte empfangen
009D:	0157	Nachrichten Flag für Kernal
009E:	0158	Kassettenfehler Pass 1: Zeichenfehler
009F:	0159	Kassettenfehler Pass 2: korrigiert
00A0:	0160 - 0162	24 Stunden Echtzeit Uhr : 1/60 Sekunden Einheiten
00A3:	0163 - 0164	Zwischenspeicher für seriellen Bus
00A5:	0165	Abwärtszähler beim SAVE auf Band, ser. Hilfszeiger
00A6:	0166	Zeiger für Kassettenpuffer
00A7:	0167	Band Kurzzähler, RS232 Eingabebits
00A8:	0168	Band Lesefehler, RS232 Zähler Eingabebits
00A9:	0169	Band 0 Leseflag, RS232 Startbit Flag
00AA:	0170	Band READ Modus, RS232 Puffer Eingabebyte
00AB:	0171	Band Kurzzähler, RS232 Eingabeparität
00AC:	0172 - 0173	Zeiger: Bildschirm scrollen, Kassettenpuffer Lo/Hi
00AE:	0174 - 0175	Zeiger: Programmende, Kassettenende Lo/Hi
00B0:	0176 - 0177	Kassetten Konstante für Zeit
00B2:	0178 - 0179	Zeiger: Anfang des Kassetten Puffers Lo/Hi
00B4:	0180	Band Hilfszeiger, RS232 nächstes Bit für scrollen
00B5:	0181	Band EOT Zeichen, RS232 nächstes Bit für übertr.
00B6:	0182	Band Hilfszeiger, RS232 Bytepuffer
00B7:	0183	Länge des aktuellen Dateinamens
00B8:	0184	Logische Dateinummer (LFN)
00B9:	0185	Aktuelle Sekundäradresse (SA)
00BA:	0186	Aktuelle Geräteadresse (GA)
00BB:	0187 - 0188	Zeiger: Adresse des Aktuellen Dateinamens Lo/Hi
00BD:	0189	Band Hilfszeiger, RS232 rotieren Paritäts Puffer
00BE:	0190	Zahl der zum Lesen/Schreiben verbliebenen Blöcke
00BF:	0191	Serieller Puffer
00C0:	0192	Flag: Kassettenmotor
00C1:	0193	Startadresse für Ein- Ausgabe (Lo), Track Nr.
00C2:	0194	Startadresse für Ein- Ausgabe (Hi), Sektor Nr.
00C3:	0195 - 0196	Band LOAD temporär, Zeiger Kernal Vektoradressen
00C5:	0197	Band Lesen/Schreiben Datenbereich
00C6:	0198	Bank Nr. für aktuelle LOAD, SAVE, VERIFY Aufrufe
00C7:	0199	Bank Nr. wo aktueller Filename unter $BB,$BC steht
00C8:	0200 - 0201	Zeiger: RS232 Eingabepuffer

00CC:	0204 - 0205	Zeiger: Tastatur Dekodiertabelle
00CE:	0206 - 0207	Zeiger auf String Pos. für Kernal PRIMM Routine
00D0:	0208	Index auf Tastaturpuffer Warteschlamge
00D1:	0209	Flag für Funktionstaste bei Aufruf
00D2:	0210	Index auf den Funktionstastenstring bei Aufruf
00D3:	0211	Sh-Flag:Shift=$01, C=$02, Ctrl=$04, Alt=$08, DIN=$10
00D4:	0212	Flag für eine gedrückte Taste
00D5:	0213	Flag für momentan gedrückte Taste (CHR$(0)=keine)
00D6:	0214	Flag für INPUT oder GET über Tastatureingabe
00D7:	0215	Flag für 40 / 80 Zeichen Modus (40=$00. 80=$80)
00D8:	0216	Flag für Text- Graphikbildschirm Modus
00D9:	0217	Zeiger Zeichensatz im RAM oder ROM (0=ROM, 4=RAM)
00DA:	0218	Zeiger für MOVLIN (Lo), F-Taste Stringlänge
00DB:	0219	Zeiger für MOVLIN (Hi), Stringlänge aller F-Tasten
00DC:	0220	Nummer der Funktionstaste
00DD:	0221	F-Tasten Stringlänge bis zur aktuellen F-Taste
00DE:	0222	Bank für Funktionstastenaufruf <sedt1>
00DF:	0223	F-Tasten Stringlänge bis zur aktuellen (F-Taste-1)
00E0:	0224 - 0225	Zeiger auf laufende Bildschirmzeile: Text Ram
00E2:	0226 - 0227	Zeiger auf laufende Bildschirmzeile: Attribut Ram
00E4:	0228	Untere Grenze des Fensters
00E5:	0229	Obere Grenze des Fensters
00E6:	0230	Linke Grenze des Fensters
00E7:	0231	Rechte Grenze des Fensters
00E8:	0232	Start der laufenden Eingabezeile
00E9:	0233	Start der laufenden Eingabespalte
00EA:	0234	Ende der laufenden Eingabezeile
00EB:	0235	Aktuelle Cursorposition: Zeile
00EC:	0236	Aktuelle Cursorposition: Spalte
00ED:	0237	Maximale Anzahl der Bildschirmzeilen
00EE:	0238	Maximale Anzahl der Bildschirmspalten
00EF:	0239	Zwischenspeicher für auszugebendes Zeichen
00F0:	0240	Speicher: vorangegangenes Zeichen (für ESC Test)
00F1:	0241	Aktueller Farbcode unter Cursor für Zeichenausgabe
00F2:	0242	Farbcode Sicherung für INSERT und DELETE
00F3:	0243	Flag: Rervers (RVS) Modus aktiv
00F4:	0244	Flag: Anführungszeichen (Quote) Modus aktiv
00F5:	0245	Flag: Einfüge (Insert) Modus aktiv
00F6:	0246	Flag: Automatisches Einfügen (Auto Insert) aktiv
00F7:	0247	Umschaltverriegelung C-Shift ($80) u. Ctrl-S ($40)
00F8:	0248	Verriegeln des Bildschirm scrollens
00F9:	0249	Verriegeln des mittels Ctrl-G erzeugten Beep-Tones
00FA:	0250 - 0254	Freier Bereich für Benutzeranwendungen
00FF:	0255	Lofbuf

****

0100:	0256 - 0271	16 Byte Bereich zur Bildung von Dateinamen

0100:	0256 - 0271	16 Byte Bereich zur Bildung von Dateinamen	
0110:	0272	DOS Schleifenzähler	
0111:	0273	DOS Länge des 1. Dateinamens	
0112:	0274	DOS Geräteadresse 1. Diskettenlaufwerk	
0113:	0275 - 0276	DOS Adresse des 1. Dateinamens	Lo/Hi
0115:	0277	DOS Länge des 2. Dateinamens	
0116:	0278	DOS Geräteadresse 2. Diskettenlaufwerk	
0117:	0279 - 0280	DOS Adresse des 2. Dateinamens	Lo/Hi
0119:	0281 - 0282	Anfangsadresse für BLOAD / BSAVE Befehl	Lo/Hi
011B:	0283 - 0284	Endadresse für BSAVE Befehl	Lo/Hi
011D:	0285	DOS Logische Adresse	
011E:	0286	DOS Physikalische Adresse	
011F:	0287	DOS Sekundäradresse	
0120:	0288	DOS Länge eines Datensatzes	
0121:	0289	DOS BANK Nummer	
0122:	0290 - 0291	DOS 2 Byte Speicher für Disketten ID	
0124:	0292	DOS Flag für Disk-ID Prüfung	
0125:	0293	PRINT USING Zeiger auf Anfangsnummer	
0126:	0294	PRINT USING Zeiger auf Endenummer	
0127:	0295	PRINT USING Flag für Dollar ($) Zeichen	
0128:	0296	PRINT USING Flag für Komma (,) Zeichen	
0129:	0297	PRINT USING Zähler	
012A:	0298	PRINT USING Vorzeichen des Exponenten	
012B:	0299	PRINT USING Zeiger auf den Exponenten	
012C:	0300	PRINT USING Zähler für Anzahl der Vorkommastellen	
012D:	0301	PRINT USING Flag für Ausrichtung nach Dezimalpunkt	
012E:	0302	PRINT USING Zähler Feldpositionen vor Dezimalpunkt	
012F:	0303	PRINT USING Zähler Feldpositionen nach Dezimalpunkt	
0130:	0304	PRINT USING Flag für Vorzeichenfeld (+/-)	
0131:	0305	PRINT USING Flag für Exponent des Feldes	
0132:	0306	PRINT USING Schalter	
0133:	0307	PRINT USING Zähler für die Zeichen eines Feldes	
0134:	0308	PRINT USING Vorzeichennummer	
0135:	0309	PRINT USING Flag für Blank oder Sternchen (/*)	
0136:	0310	PRINT USING Zeiger auf den Beginn des Feldes	
0137:	0311	PRINT USING Zeiger für Länge des Formats	
0138:	0312	PRINT USING Zeiger auf das Ende des Feldes	

****

0139:	0313 - 0510	Ende des System-Stacks
01FF:	0511	Anfang des System-Stacks

****

0200:	0512	Basic und Monitor Eingabe-Puffer

```
***********************************************************************

02A2:  0674        FETCH Routine: LDA(ZP),Y von beliebiger Bank

02A2:  AD 00 FF    LDA  $FF00     Eine genaue Beschreibung
02A5:  8E 00 FF    STX  $FF00     dieser Routine finden Sie im
02A8:  AA          TAX            ROM Listing unter $F800, da
02A9:  B1 FF       LDA  ($FF),Y   dort die ROM Kopie zu finden ist.
02AB:  8E 00 FF    STX  $FF00     Der "FETVEC" hat die Adresse:
02AE:  60          RTS            $02AA, bzw. Dez. 0682.

***********************************************************************

02AF:  0687        STASH Routine: STA(ZP),Y in beliebige Bank

02AF:  48          PHA            Eine genaue Beschreibung
02B0:  AD 00 FF    LDA  $FF00     dieser Routine finden Sie im
02B3:  8E 00 FF    STX  $FF00     ROM Listing unter $F80D, da
02B6:  AA          TAX            dort die ROM Kopie zu finden ist.
02B7:  68          PLA            Der "STAVEC" hat die Adresse:
02B8:  91 FF       STA ($FF),Y    $02B9, bzw. Dez. 0697.
02BA:  8E 00 FF    STX  $FF00
02BD:  60          RTS

***********************************************************************

02BE:  0702        CMPARE Routine: CMP(ZP),Y mit beliebiger Bank

02BE:  48          PHA            Eine genaue Beschreibung
02BF:  AD 00 FF    LDA  $FF00     dieser Routine finden Sie im
02C2:  8E 00 FF    STX  $FF00     ROM Listing unter $F81C, da
02C5:  AA          TAX            dort die ROM Kopie zu finden ist.
02C6:  68          PLA            Der "CMPVEC" hat die Adresse:
02C7:  D1 FF       CMP  ($FF),Y   $02C8, bzw. Dez. 0712.
02C9:  8E 00 FF    STX  $FF00
02CC:  69          RTS

***********************************************************************

02CD:  0717        JSRFAR Routine: JSR in beliebige Bank und Return

02CD:  20 E3 02    JSR  $02E3     Eine genaue Beschreibung
02D0:  85 06       STA  * $06     dieser Routine finden Sie im
02D2:  86 07       STX  * $07     ROM Listing unter der
02D4:  84 08       STY  * $08     Adresse $F82B, da dort die
02D6:  08          PHP            ROM Kopie dieser Routine steht.
02D7:  68          PLA
02D8:  85 05       STA  * $05
```

```
02DA:  BA          TSX
02DB:  86 09       STX  * $09
02DD:  A9 00       LDA  # $00
02DF:  8D 00 FF    STA  $FF00
02E2:  60          RTS

***************************************************************************

02E3:  0739        JMPFAR Routine: JMP in beliebige Bank ohne Return

02E3:  A2 00       LDX  # $00      Eine genaue Beschreibung
02E5:  B5 03       LDA  * $03,X    dieser Routine finden Sie im
02E7:  48          PHA             ROM Listing unter der
02E8:  E8          INX             Adresse $F841, da dort die
02E9:  E0 03       CPX  # $03      ROM Kopie dieser Routine zu
02EB:  90 F8       BCC  $02E5      finden ist.
02ED:  A6 02       LDX  * $02
02EF:  20 6B FF    JSR  $FF6B
02F3:  8D 00 FF    STA  $FF00
02F6:  A5 06       LDA  * $06
02F8:  A6 07       LDX  * $07
02FA:  A4 08       LDY  * $08
02FB:  40          RTI

***************************************************************************
```

Routine zum Sprung in eine Funktions-Cartridge. Der Cartridge-Vektor hat die Adresse: $02FE-$02FF (Dez. 766-767)

```
02FC:  78          SEI             Alle System-Interrupts verhindern
02FD:  4C 00 00    JMP  $0000      Sprung an Funktions-Cartridge Vek.

***************************************************************************

0300:  0768   3F 4D   ($4D3F)   Vektor: Fehler Routine (X = Fehler)
0302:  0770   C6 4D   ($4DC6)   Vektor: Einles./Ausführen Bas.Zeile
0304:  0772   0D 43   ($430D)   Vektor: Umwandlung  Interpretercode
0306:  0774   51 51   ($5151)   Vektor: Umwandlg in Klartext (List)
0308:  0776   A2 4A   ($4AA2)   Vektor: Ausführen d. Schlüsselworte
030A:  0778   DA 78   ($78DA)   Vektor: Ausdruck auswerten
030C:  0780   21 43   ($4321)   Vektor: Escape Umwandlungsroutine
030E:  0782   CD 51   ($51CD)   Vektor: Escape List
0310:  0784   A9 4B   ($4BA9)   Vektor: Escape ausführen
0312:  0786   FF FF   ($FFFF)   Interruptvektor: TIME
0314:  0788   65 FA   ($FA65)   Vektor für IRQ Routine
0316:  0790   03 B0   ($B003)   Vektor für Break Einsprung (Monitor)
0318:  0792   40 FA   ($FA40)   Vektor für NMI Routine
031A:  0794   BD EF   ($EFBD)   Vektor auf Kernal OPEN  Routine
```

```
031C:  0796       88 F1   ($F188)   Vektor auf Kernal CLOSE Routine
031E:  0798       06 F1   ($F106)   Vektor auf Kernal CHKIN Routine
0320:  0800       4C F1   ($F14C)   Vektor auf Kernal CKOUT Routine
0322:  0802       26 F2   ($F226)   Vektor auf Kernal CLRCH Routine
0324:  0804       06 EF   ($EF06)   Vektor auf Kernal BASIN Routine
0326:  0806       79 EF   ($EF79)   Vektor auf Kernal BSOUT Routine
0328:  0808       6E F6   ($F66E)   Vektor auf Kernal STOP  Routine
032A:  0810       EB EE   ($EEEB)   Vektor auf Kernal GETIN Routine
032C:  0812       22 F2   ($F222)   Vektor auf Kernal CLALL Routine
032E:  0814       06 B0   ($B006)   Vektor auf EXMON Einsprung
0330:  0816       6C F2   ($F26C)   Vektor auf Kernal LOAD Routine
0332:  0818       4E F5   ($F54E)   Vektor auf Kernal SAVE Routine

**********************************************************************

Kopie der Zeichenausgabe-, Tastatur- und Decodiervektoren
Die Originale dieser Vektoren stehen im ROM von Adr. $C065 - $C07A

0334:  0820       B9 C7   ($C7B9)   Vektor für Zeichenausgabe mit Ctrl
0336:  0822       05 C8   ($C805)   Vektor für Zeichenausgabe mit Shift
0338:  0824       C1 C9   ($C9C1)   Vektor für Zeichenausgabe mit Esc
033A:  0826       E1 C5   ($C5E1)   Vektor für Tastaturabfrage
033C:  0828       AD C6   ($C6AD)   Vektor auf Tastendruck speichern
033E:  0830       80 FA   ($FA80)   Vektor: Tastatur Decodiertabelle 1a
0340:  0832       D9 FA   ($FAD9)   Vektor: Tastatur Decodiertabelle 2a
0342:  0834       32 FB   ($FB32)   Vektor: Tastatur Decodiertabelle 3a
0344:  0836       8B FB   ($FB8B)   Vektor: Tastatur Decodiertabelle 4a
0346:  0838       80 FA   ($FA80)   Vektor: Tastatur Decodiertabelle 1a
0348:  0840       E4 FB   ($FBE4)   Vektor: Tastatur Decodiertabelle 5a

**********************************************************************

034A:  0842 - 0851  IRQ Tastatur Puffer
0354:  0852 - 0861  Bitmuster Tabelle: Tabulator Stops
035E:  0862 - 0865  Bitmuster Tabelle: Zeilen Übergänge
0362:  0866 - 0875  Tabelle der logischen Dateinummern
036C:  0876 - 0885  Tabelle der Geräteadressen
0376:  0886 - 0895  Tabelle der Sekundäradressen

**********************************************************************

0380:  0896         Basic CHRGET Routine
                    (Das Original steht im ROM an Adresse $4279)

0380:  E6 3D        INC  * $3D   BASIC-Textzeiger Lo erhöhen
0382:  D0 02        BNE  $0386   Kein Überlauf, dann Skip
0384:  E6 3E        INC  * $3E   BASIC-Textzeiger Hi erhöhen
```

```
***********************************************************************

0386:  0902        Basic CHRGOT Routine
                   (Das Original steht im ROM an Adresse $427F)

0386:  8D 01 FF    STA  $FF01     RAM 0 Bereich einschalten
0389:  A0 00       LDY  # $00     Displacementzeiger auf BASIC-Text
038B:  B1 3D       LDA  ($3D),Y   Hole ein Zeichen aus BASIC-Text
038D:  8D 03 FF    STA  $FF03     RAM 0, System ROMs einschalten

***********************************************************************

0390:  0912        Basic QNUM Routine
                   setzt Zero-Flag bei Trennzeichen $00 oder $3A
                   setzt Carry-Flag bei Ziffernzeichen 0 - 9
                   (Das Original steht im ROM an Adresse $4289)

0390:  C9 3A       CMP  # $3A     Zeichencode größer als Zifferncode?
0392:  B0 0A       BCS  $039B     Ja, dann Skip
0394:  C9 20       CMP  # $20     War gelesenes Zeichen ein "Blank"?
0396:  F0 EB       BEQ  $0380     Ja, dann Leerzeichen überlesen
0398:  38          SEC            Carry für Subtraktion setzen
0399:  E9 30       SBC  # $30     Test auf Ziffernzeichen (dann C = 1)
039B:  38          SEC            Carry für Subtraktion setzen
039C:  E9 D0       SBC  # $D0     Stelle alten Wert wieder her
039E:  60          RTS            Rücksprung aus dem Unterprogramm

***********************************************************************

039F:  0927        Laden aus beliebiger Bank über PCRA und PCRC
                   (Das Original steht im ROM an Adresse $4298)

039F:  8D A6 03    STA  $03A6
03A2:  8D 01 FF    STA  $FF01
03A5:  B1 00       LDA  ($00),Y
03A7:  8D 03 FF    STA  $FF03
03AA:  60          RTS

***********************************************************************

03AB:  0939        Laden aus beliebiger Bank über PCRB und PCRD
                   (Das Original steht im ROM an Adresse $42A4)

03AB:  8D B2 03    STA  $03B2
03AE:  8D 02 FF    STA  $FF02
03B1:  B1 00       LDA  ($00),Y
03B3:  8D 04 FF    STA  $FF04
03B6:  60          RTS
```

```
***********************************************************************

03B7:  0951          Laden aus bel. Bank über PCRA und PCRC von
                     der durch Z-Page Index 1 angegebenen Adresse
                     Das Original steht im ROM an Adresse $42B0)

03B7:  8D 02 FF      STA  $FF02
03BA:  B1 24         LDA  ($24),Y
03BC:  8D 04 FF      STA  $FF04
03BF:  60            RTS

***********************************************************************

03C0:  0960          Laden aus bel. Bank über PCRB und PCRD von
                     der durch Z-Page Index 2 angegebenen Adresse
                     (Das Original steht im ROM an Adresse $42B9)

03C0:  8D 01 FF      STA  $FF01
03C3:  B1 26         LDA  ($26),Y
03C5:  8D 03 FF      STA  $FF03
03C8:  60            RTS

***********************************************************************

03C9:  0969          Laden aus bel. Bank über PCRA und PCRC von
                     der durch Z-Page CHRGET Zeiger angegebenen Adresse
                     Das Original steht im ROM an Adresse $42C2)

03C9:  8D 01 FF      STA  $FF01
03CC:  B1 3D         LDA  ($3D),Y
03CE:  8D 03 FF      STA  $FF03
03D1:  60            RTS

***********************************************************************

03D2:  0978 - 0980   Numerische Konstante für Basic, v. Rom geladen
03D5:  0981          Bank für SYS,POKE,PEEK. Gesetzt durch Bank Befehl
03D6:  0982 - 0985   Zwischenspeicher für Instring
03DA:  0986          Bank Zeiger für Strings und Zahlenumwandlung
03DB:  0987 - 0990   4 Byte Arbeitsspeicher für SSHAPE Operationen
03DF:  0991          Überlauf Kennzeichen des FAC1
03E0:  0992          Zwischenspeicher zur Spritespeicherung Nr.1
03E1:  0993          Zwischenspeicher zur Spritespeicherung Nr.2
03E2:  0994          Gepackte Vodergrund/Hintergrund Farbnibbles
03E3:  0995          Gepackte Vordergrund/Multicolor Farbnibbles
03E4:  0996 - 1007   Freier Bereich
```

```
*********************************************************************

03F0:  1008          DMA-Call Routine im unteren Common Bereich (1.k)
                     zur Initialisierung des externen Speicherzugriffs

03F0:  AE 00 FF      LDX  $FF00   Eine genaue Beschreibung dieser
03F3:  8C 01 DF      STY  $DF01   DMA-Call Routine für die Kontrolle
03F6:  8D 00 FF      STA  $FF00   des externen Speicherzugriffs
03F9:  8E 00 FF      STX  $FF00   finden Sie im ROM unter d. Original-
03FC:  60            RTS          adresse $F85A

*********************************************************************

03FD:  1021 - 1023   Freier Bereich

03FF:  1023          Ende der Common-Area,die in allen Banks gleich ist

*********************************************************************

0400:  1024 - 2047   Bildschirmspeicher

*********************************************************************

0800:  2048 - 2559   512 Bytes für Basic Laufzeitspeicher

*********************************************************************

0A00:  2560 - 2561   Vektor System restart (Norm. Warmstart) ($4003)
0A02:  2562          Kernal Warm-/Kaltstart Initialisierungs Status
0A03:  2563          PAL / NTSC  System Zeiger  ($FF=PAL, $00=NTSC)
0A04:  2564          System Zeiger für den NMI und RESET Status
0A05:  2565 - 2566   Untere Grenze d. verfügbaren RAM in System Bank
0A07:  2567 - 2568   Obere Grenze des verfügbaren RAM in System Bank
0A09:  2569 - 2570   Indirekter IRQ Vektor für Kassettenroutinen
0A0B:  2571          Zeitvergleich bei Kassettenroutinen
0A0C:  2572          Zwischenspeicher beim Lesen von Kassette
0A0D:  2573          Zwischenspeicher beim Lesen von Kassette
0A0E:  2574          Timeout Zeiger für schnellen seriellen Modus
0A0F:  2575          RS232 NMI Status Register
0A10:  2576          RS232 Kontrollregister
0A11:  2577          RS232 Kommandoregister
0A12:  2578 - 2579   RS232 Benutzer Baudrate
0A14:  2580          RS232 Statusregister
0A15:  2581          RS232 Anzahl der zu sendenden Bits
0A16:  2582 - 2583   RS232 Baud Rate: Full Bit Time (in us)
0A18:  2584          RS232 Index auf den Anfang des Eingabepuffers
0A19:  2585          RS232 Index auf das Ende des Eingabepuffers
0A1A:  2586          RS232 Index auf den Anfang des Ausgabepuffers
```

0A1B:	2587	RS232 Index auf das Ende des Ausgabepuffers
0A1C:	2588	Intern/Extern Zeiger für schnellen seriellen Modus
0A1D:	2589 - 2591	Zwischenspeicher für die 24h Echtzeituhr
0A20:	2592	Speicher für die Größe des Tastaturpuffers
0A21:	2593	Pause Zeiger, <Crtl - S> Zeiger
0A22:	2594	Zeiger: Tastenwiederholungen
0A23:	2595	Zählgeschwindigkeit für Tastenwiederholungen
0A24:	2596	Zähler für Tasten Wiederholungsverzögerung
0A25:	2597	Speicher für letztes Shift Muster der Tastatur
0A26:	2598	Zeiger für Cusor in Blinkphase
0A27:	2599	Zeiger für Cursor ein/aus (0 = blinkender Cursor)
0A28:	2600	Zählzeiger für blinkenden Cursor
0A29:	2601	Zeichen für Cursorposition
0A2A:	2602	Speicher für Hintergrundfarbe unter Cursor
0A2B:	2603	Zeiger für aktuellen Cursor Modus (wenn verfügbar)
0A2C:	2604	Text Bildschirm/Zeichen Basis Zeiger
0A2D:	2605	Bit-Map Basis Zeiger
0A2E:	2606	Zeiger für Adresse (*256) für 80 Zeichen Video RAM
0A2F:	2607	Zeiger für Adresse (*256) für Attribut RAM
0A30:	2608	Hilfszeiger auf letzte Zeile für LOOP4 Routine
0A31:	2609	Zwischenspeicher (a) für 80 Zeichen Routinen
0A32:	2610	Zwischenspeicher (b) für 80 Zeichen Routinen
0A33:	2611	Zwischenspeicher (a) für Zeile löschen / verschieben
0A34:	2612	Zwischenspeicher (b) für Zeile löschen / verschieben
0A35:	2613	Farbe unter 80 Zeichen Cursor vor dem Blinken
0A36:	2614	Rasterzeile an d. Rasterinterrupt ausgelöst wird
0A37:	2615	Speicher für das X-Register bei BANK Operationen
0A38:	2616	Zähler für das PAL System, Jiffie Abgleich
0A39:	2617	Hilfsspeicher für 80 Zeichen VDC Bildschirm

****

Sicherungsspeicher für passiven Bildschirm Variablen. Dieser Bereich korrespondiert mit dem Zero-Page Bereich ab $E0.

0A40:	2624 - 2625	Zeiger auf laufende Bildschirmzeile: Text RAM	
0A42:	2626 - 2627	Zeiger auf laufende Bildschirmzeile: Attribut RAM	
0A44:	2628	Untere Grenze des Fensters	(init: $18 = 24)
0A45:	2629	Obere Grenze des Fensters	(init: $00 = 00)
0A46:	2630	Linke Grenze des Fensters	(init: $00 = 00)
0A47:	2631	Rechte Grenze des Fensters	(init: $4F = 79)
0A48:	2632	Start der laufenden Eingabezeile	(init: $00 = 00)
0A49:	2633	Start der laufenden Eingabespalte	(init: $00 = 00)
0A4A:	2634	Ende der laufenden Eingabezeile	(init: $00 = 00)
0A4B:	2635	Aktuelle Cursorposition: Zeile	(init: $00 = 00)
0A4C:	2636	Aktuelle Cursorposition: Spalte	(init: $00 = 00)
0A4D:	2637	Maximale Anzahl Bildschirmzeilen	(init: $18 = 24)
0A4E:	2638	Maximale Anzahl Bildschirmspalten	(init: $4F = 79)

0A4F:	2639	Zwischenspeicher für auszugebendes Zeichen
0A50:	2640	Speicher: Vorangegangenes Zeichen (für ESC Test)
0A51:	2641	Aktueller Farbcode unter Cursor (init: $07 = 07)
0A52:	2642	Farbcode Sicherung (Insert+Delete)(init: $07 = 07)
0A53:	2643	Zeiger für RVS Modus aktiv
0A54:	2644	Zeiger für Anführungszeichen (Quote) Modus aktiv
0A55:	2645	Zeiger für Einfüge (Insert) Modus aktiv
0A56:	2646	Zeiger für automatisches Einfügen aktiv
0A57:	2647	Zeiger für Umschaltverriegelung und Pause Zeiger
0A58:	2648	Zeiger für Verriegeln des Bildschirm scrollens
0A59:	2649	Zeiger für Verriegeln des Beep Tones (Ctrl-G)

****

0A60:	2650 - 2687	Hilfsspeicherbereich für 40 und 80 Zeichen
0A80:	2688 - 2719	Buffer für Vergleichsoperationen
0AA0:	2720 - 2729	Hilfszähler
0AAA:	2730	Adressierungsart für Assembler Befehl
0AAB:	2731	Länge des Befehlscodes für Assembler/Disassembler
0AAC:	2732 - 2734	Assembler/Disassembler Speicher f. integ. Monitor
0AAF:	2735	Ein Byte Zwischenspeicher für verschiedene Zwecke
0AB0:	2736	Ein Byte Zwischenspeicher für verschiedene Zwecke
0AB1:	2737	Ein Byte Zwischenspeicher für verschiedene Zwecke
0AB2:	2738	X-Reg Speicher b. indirekten Unterprogrammaufrufen
0AB3:	2739	Richtungszeiger für Transfer Operationen
0AB4:	2740 - 2751	Ein Byte Zwischenspeicher
0AC0:	2752	ROM Bank für aktuellen Funktionstastenaufruf
0AC1:	2753 - 2756	Tabelle physikalischer Adressen und ID's von eingesteckten Erweiterungskarten
0AC5:	2757	System Zeiger für die Zusammensetzung von Vokalen mit Akzenten im DIN-Zeichensatz

****

0B00:	2816 - 3071	Kassettenpuffer

****

0C00:	3072 - 3327	RS232 Eingabe Puffer

****

0D00:	3328 - 3583	RS232 Ausgabe Puffer

****

0E00:	3584 - 4095	Bereich z. Spritedefinition (muß unter $1000 sein)

```
*********************************************************************

1000:  4096 - 4105  Programmierbare Funktionstasten (Längentabelle)
100A:  4106 - 4351  Programmierbare Funktionstasten (Funktionsstrings)

*********************************************************************

1100:  4352 - 4400  Puffer zur Bildung v. DOS Ausgabe Strings
1131:  4401 - 4402  Grafik Variable: Aktuelle X-Position       (Lo/Hi)
1133:  4403 - 4404  Grafik Variable: Aktuelle Y-Position       (Lo/Hi)
1135:  4405 - 4406  Grafik Variable: Zielrichtung X-Koordinate (Lo/Hi)
1137:  4407 - 4408  Grafik Variable: Zielrichtung Y-Koordinate (Lo/Hi)
1139:  4409 - 4410  Variable für Grafik Linien: X/Y-Absolut, X-Absolut
113B:  4411 - 4412  Variable für Grafik Linien: Y-Absolut
113D:  4413 - 4414  Variable für Grafik Linien: X/Y-Signum , X-Signum
113F:  4415 - 4416  Variable für Grafik Linien: Y-Signum
1141:  4417 - 4420  Variable für Grafik Linien: Faktor
1145:  4421 - 4422  Variable für Grafik Linien: Fehlerwert
1147:  4423         Variable für Grafik Linien: Kleiner Kennzeichen
1148:  4424         Variable für Grafik Linien: Größer Kennzeichen
1149:  4425         Variable für Winkel Rout.: Vorzeichen des Winkels
114A:  4426 - 4427  Variable für Winkel Rout.: Sinus des Winkelwertes
114C:  4428 - 4429  Variable für Winkel Rout.: Cosinus d Winkelwertes
114E:  4430 - 4431  Variable für Winkel Rout.: Winkel Distanz

*********************************************************************
Die folgenden 24 Byte sind für verschiedene Zwecke mehrfach belegt

                    Variablen für Kreis Routinen

1150:  4432 - 4433  Kreis Mittelpunkt: X-Koordinate            (Lo/Hi)
1152:  4434 - 4435  Kreis Mittelpunkt: Y-Koordinate            (Lo/Hi)
1154:  4436 - 4437  Kreis Radius in X-Richtung                 (Lo/Hi)
1156:  4438 - 4439  Kreis Radius in Y-Richtung                 (Lo/Hi)
1158:  4440 - 4443  Rotationswinkel des Kreises                (Lo/Hi)
115C:  4444 - 4445  Winkelgrad für Anfang des Kreisbogens      (Lo/Hi)
115E:  4446 - 4447  Winkelgrad für Ende des Kreisbogens        (Lo/Hi)
1160:  4448 - 4449  X-Radius * Cos(Rotationswinkel)
1162:  4450 - 4451  Y-Radius * Sin(Rotationswinkel)
1164:  4452 - 4453  X-Radius * Sin(Rotationswinkel)
1166:  4454 - 4455  Y-Radius * Cos(Rotationswinkel)

                    Mehrfach benutzte Parameter für allgemeine Zwecke

1150: 4432 - 4433   Mittelpunkt für X-Koordinate
1152: 4434 - 4435   Mittelpunkt für Y-Koordinate
1154: 4436 - 4437   Abstand 1 für X-Koordinate
1156: 4438 - 4439   Abstand 1 für Y-Koordinate
```

```
1158: 4440 - 4441   Abstand 2 für X-Koordinate
115A: 4442 - 4443   Abstand 2 für Y-Koordinate
115C: 4444 - 4445   Koordinaten Abstandsende
115E: 4446          Spaltenzähler für Zeichen
115F: 4447          Zeilenzähler  für Zeichen
1160: 4448:         Längenzähler  für Zeichenkette

                    Mehrfach benutzte Variablen für Rechteck Routinen

1150: 4432 - 4433   X-Koordinate 1
1152: 4434 - 4435   Y-Koordinate 1
1154: 4436 - 4437   Rotationswinkel
1156: 4438 - 4439   Zähler für X-Wert
1158: 4440 - 4441   Zähler für Y-Wert
115A: 4442 - 4443   Länge einer Seite des Rechtecks
115C: 4444 - 4445   X-Koordinate 2
115E: 4446 - 4447   Y-Koordinate 2

                    Mehrfach benutzt für Shapes und Shape Bewegung

1150: 4432          Platzhalter
1151: 4433          Längenzeiger
1152: 4434          Folgezeiger
1153: 4435          Länge der Zeichenkette
1154: 4436          Shape-Modus setzen / ersetzen
1155: 4437          Zeiger auf Position in der Zeichenkette
1156: 4438          Altes Bit-Map Byte
1157: 4439          Variable f. neuen String oder Bit-Map Byte
1158: 4440          Platzhalter
1159: 4441 - 4442   Spaltenbreite (X-Breite) eines Shapes
115B: 4443 - 4444   Zeilenzahl (Y-Länge) eines Shapes
115D: 4445 - 4446   Zwischenspeicher für die Spaltenbreite
115F: 4447 - 4448   Zeiger auf Shape-String bei der Shape Speicherung
1161: 4449          Bitzeiger auf Byte des Shape Strings

**********************************************************************

                    Bereich für allgemeine Grafik Variablen

1168: 4456          Hilfsspeicher für diverse Zwecke
1169: 4457          Hilfsspeicher: Bitzähler für GSHAPE Anweisung
116A: 4458          Bildschirm Skalierungszeiger 0=320*200,1=1024*1024
116B: 4459          Hilfszeiger für doppelte Breite
116C: 4460          Hilfszeiger für Rechteck ausfüllen (Box fill)
116D: 4461          Hilfsspeicher für Bitmasken
116E: 4462          Hilfszähler für numerische Werte
116F: 4463          Hilfszeiger für Trace Modus ein/aus
1170: 4464 - 4465   Hilfsspeicher 1 für Renumber Routine
```

1172:	4466 - 4467	Hilfsspeicher 2 für Renumber Routine
1174:	4468	1 Byte Hilfsspeicher
1175:	4469 - 4470	2 Byte Hilfsspeicher
1177:	4471	1 Byte Hilfsspeicher 1 für Grafikroutinen
1178:	4472	1 Byte Hilfsspeicher 2 für Grafikroutinen
1179:	4473	1 Byte Hilfsspeicher 3 für Grafikroutinen
117A:	4474 - 4475	Vektor: Umwandlung Fließpunkt in Integer ($849F)
117C:	4476 - 4477	Vektor: Umwandlung Integer in Fließpunkt ($793C)
117E:	4478 - 4565	Geschwindigkeits-/Richtungstabelle für Sprites
11D6:	4566 - 4607	42 Byte Bereich für Kopie der VIC Register
1200:	4608 - 4609	Vorherige Basic Zeilennummer
1202:	4610 - 4611	Befehlszeiger für Basic CONTinue Befehl
1204:	4612	Print Using Zeiger: Chr$
1205:	4613	Print Using Zeiger: Füllzeichen
1206:	4614	Print Using Zeiger: Kommazeichen
1207:	4615	Print Using Zeiger: Zeichen für Dezimalpunkt
1208:	4616	Letzte aufgetretene Fehlernummer (für TRAP Befehl)
1209:	4617 - 4618	Zeilennummer des letzten Fehlers ($FFFF ist OK KZ)
120B:	4619 - 4620	Zeilennummer,die bei Fehler ausgeführt werden soll
120D:	4621	Hilfszeiger für TRAP Befehl
120E:	4622 - 4623	Zeiger auf Text der Fehlermeldung
1210:	4624 - 4625	Textendezeiger
1212:	4626 - 4627	Höchste für Basic verfügbare Adresse in RAM 0
1214:	4628 - 4629	Hilfsspeicher für DO - LOOP
1216:	4630 - 4631	Hilfsspeicher für Zeilennummer
1218:	4632	USR Sprung
1219:	4633 - 4634	USR Adresse im Format Lo/Hi
121B:	4635 - 4639	Eingangswert der RND Funktion
1220:	4640	Gradzahl für ein Kreissegment
1221:	4641	Zeiger auf Reset Status (Kaltstart oder Warmstart)

****

Speicherbereich für Musik Zeiger

1222:	4642	<tempo rate>
1223:	4643 - 4648	<voices>
1229:	4649 - 4650	<ntime>
122B:	4651	<octave>
122C:	4652	<sharp>
122D:	4653 - 4654	<pitch>
122F:	4655	<voice>
1230:	4656 - 4658	<wave 0>
1233:	4659	<dnote>
1234:	4660 - 4663	<fltsav>
1238:	4664	<fltflg>
1239:	4665	<nibble>
123A:	4666	<tonnum>

```
123B: 4667 - 4669   <tonval>
123E: 4670          <parcnt>
123F: 4671 - 4680   <atktab>
1249: 4681 - 4690   <sustab>
1253: 4691 - 4700   <waftab>
125D: 4701 - 4710   <pulslw>
1267: 4711 - 4720   <pulshi>
1271: 4721 - 4725   <filters>

*********************************************************************

                    Speicherbereich für Interrupt Zeiger

1276: 4726 - 4728   3 Byte Interrupt Speicher
1279: 4729 - 4731   3 Byte Interrupt Adresse Lo Speicher
127C: 4732 - 4734   3 Byte Interrupt Adresse Hi Speicher
127F: 4735          <intval>
1280: 4736          <coltyp>

*********************************************************************

                    Speicherbereich für SID Variablen

1281: 4737          Sound: Voice Speicher
1282: 4738 - 4740   Sound: Zeitspeicher Lo-Wert  (3 Byte)
1285: 4741 - 4743   Sound: Zeitspeicher Hi-Wert  (3 Byte)
1288: 4744 - 4746   Sound: Maximalwert  Lo       (3 Byte)
128B: 4747 - 4749   Sound: Maxiamlwert  Hi       (3 Byte)
128E: 4750 - 4752   Sound: Minimalwert  Lo       (3 Byte)
1291: 4753 - 4755   Sound: Minimalwert  Hi       (3 Byte)
1294: 4756 - 4758   Sound: Richtung              (3 Byte)
1297: 4759 - 4761   Sound: Schrittzahl  Lo       (3 Byte)
129A: 4762 - 4764   Sound: Schrittzahl  Hi       (3 Byte)
129D: 4765 - 4767   Sound: Frequenz     Lo       (3 Byte)
12A0: 4768 - 4770   Sound: Frequenz     Hi       (3 Byte)
12A3: 4771          Zwischenspeicher: Zeitwert          Lo
12A4: 4772          Zwischenspeicher: Zeitwert          Hi
12A5: 4773          Zwischenspeicher: Maximalwert       Lo
12A6: 4774          Zwischenspeicher: Maximalwert       Hi
12A7: 4775          Zwischenspeicher: Minimalwert       Lo
12A8: 4776          Zwischenspeicher: Minimalwert       Hi
12A9: 4777          Zwischenspeicher: Richtung
12AA: 4778          Zwischenspeicher: Schrittzahl       Lo
12AB: 4779          Zwischenspeicher: Schrittzahl       Hi
12AC: 4780          Zwischenspeicher: Frequenz          Lo
12AD: 4781          Zwischenspeicher: Frequenz          Hi
12AE: 4782          Zwischenspeicher: Pulswellenbreite Lo
12AF: 4783          Zwischenspeicher: Pulswellenbreite Hi
```

```
12B0: 4784           Zwischenspeicher: Wellenform
12B1: 4785           Zwischenspeicher 1 für POT Funktion
12B2: 4786           Zwischenspeicher 2 für POT Funktion
12B3: 4787 - 4790    Zwischenspeicher für WINDOW Operationen Lo/Hi
12B7: 4791 - 4857    Speicherzeiger für SPRDEF und SAVSPR Befehle
12FA: 4858           Definit. Modus für SPRDEF und SAVSPR Befehle
12FB: 4859           Zeilenzähler   für SPRDEF und SAVSPR Befehle
12FC: 4860 - 4863    Sprite Nummer  für SPRDEF und SAVSPR Befehle

**********************************************************************

1300: 4864 - 6143    Nicht belegter absoluter RAM Bereich
1800: 6144 - 7167    Reserviert für Funktionstastenanwendungen
1C00: 7168 - 8191    Video Matrix #2 (1 kB, Bitmap Farbe) wenn benötigt

**********************************************************************

2000: 8192 -16383    VIC Bitmap (8 kB) wenn benötigt

**********************************************************************

4000: 16384          Beginn des ROM über dem RAM

**********************************************************************
```

9.13 Einbinden eigener Befehle

9.13.1 Das Befehlsmodul

Als Leser des Buchs 128 Intern sind Sie sicher auch kein Freund von halben Sachen. Deshalb wollen wir jetzt auch einen Befehl und eine Funktion voll in das BASIC 7.0 einbinden. Und zwar mit Tokenumwandlung, List- und Ausführungsmöglichkeit. Das erfordert natürlich mehr Aufwand als etwa die Benutzung der USR-Funktion für solche Zwecke. Es bringt aber hinterher die Befriedigung, daß man sich nicht mit irgendwelchen Erkennungszeichen für die Befehle herumschlagen muß. Man kann dann jeden neuen Befehl wie alle anderen benutzen und sogar alte Kommandos ohne spezielle Abfragen durch neue ersetzen. Ferner können Sie verschiedene Kommandos voneinander unterscheiden. Dabei hilft Ihnen das Befehlsmodul, das hier in mehreren Schritten erläutert wird.

Das BASIC 7.0 hat wesentlich mehr als 128 Kommandos. Deshalb hat Commodore die Sondertoken $FE und $CE eingeführt, auf die noch ein Byte folgt. Dieses zweite Byte ist die eigentliche Nummer des Befehls oder der Funktion, die aufgerufen werden soll. $FE ist die Kennung für Befehle und $CE die Kennung für Funktionen. Wenn Sie schon im ROM-Listing geblättert haben, dann haben Sie sicher im Interpreter in der Befehlsverarbeitung sogenannte Escape-Vektoren entdeckt. Sie werden benutzt, wenn der Interpreter hinter einem Sondertoken eine Kommandonummer findet, die er nicht verarbeiten kann. Und genau an dieser Stelle ist der Punkt, an dem wir einhaken.

Zuerst bringen wir dem Interpreter bei, unsere Kommandos als Sondertoken mit Befehlsnummern größer als 127 abzulegen. Unsere Assemblerprogramme für diesen Zweck legen wir in die Programmbank 0 ab $F000. Man braucht ja immer mehr Platz für Variablen, als für ein Programm. Wie bringt man den Interpreter aber dazu, daß er aus dem ROM in einen RAM-Bereich *unter* dem Betriebssystem springt? Wir benutzen dazu das Sprungmodul aus Kapitel 9.9.1 und hängen daran eine kleine Sprungtabelle für die Escape-Vektoren. Tippen Sie also zuerst das Sprungmodul und dann folgenden Speicherauszug ein:

```
>01780 20 03 17 4c 00 f0 24 00 20 03 17 4c 03 f0 24 00
>01790 20 03 17 4c 06 f0 24 00 20 03 17 4c 09 f0 24 00
>017a0 60
```

Damit Sie sich darunter auch etwas vorstellen können, folgt hier der Klartext:

```
1780   20 03 17   jsr $1703    Sprungmodul JMP-Befehl
1783   4c 00 f0   jmp $f000    Funktionsvektor auf $F000
1786   24 00      bit $00      in Bank 0
1788   20 03 17   jsr $1703    Sprungmodul JMP-Befehl
178b   4c 03 f0   jmp $f003    Tokenumwandlung auf $F003
178e   24 00      bit $00      in Bank 0
1790   20 03 17   jsr $1703    Sprungmodul JMP-Befehl
1793   4c 06 f0   jmp $f006    Listvektor auf $F006
1796   24 00      bit $00      in Bank 0
1798   20 03 17   jsr $1703    Sprungmodul JMP-Befehl
179b   4c 09 f0   jmp $f009    Befehlsvektor auf $F009
179e   24 00      bit $00      in Bank 0
17a0   60         rts          Für Rücksprung ins ROM
```

Für alles ist also ein Sprung vorhanden, für die Funktionsausführung, für die Umwandlungen in Token und in Klartext und für die Befehlsausführung. Der RTS-Befehl ist für den Rücksprung ins ROM nötig. Man kann zwar mit dem Sprungmodul direkt ins BASIC-ROM einspringen, ein Rücksprung mit RTS ist jedoch wegen der nötigen Umschaltung der Speicherkonfiguration nicht möglich. Sie werden jetzt sagen, daß man auch ein RTS im ROM hätte anspringen können. Das stimmt natürlich. Der Logik nach aber sollte dieser Befehl auch in der Sprungtabelle stehen.

Da wir uns zuerst mit der Umwandlung in Befehlstoken und wieder in Klartext befassen wollen, brauchen wir erst die Hälfte des Befehlsmoduls einzutippen:

```
>0f000 4c da f0 4c 32 f0 4c ba f0 4c 00 f1 20 03 17 4c
>0f010 78 4c 24 0f 20 03 17 4c 21 43 24 0f 20 03 17 4c
>0f020 2e 51 24 0f 20 03 17 4c a9 4b 24 0f 2c 00 f8 2c
>0f030 00 f9 ac 2d f0 ad 2e f0 20 62 f0 90 06 a2 00 a5
>0f040 0d d0 0f ac 30 f0 ad 31 f0 20 62 f0 90 08 a2 ff
```

```
>0f050 a5 0d 18 4c 14 f0 20 00 17 20 86 03 24 00 38 4c
>0f060 14 f0 85 25 84 24 a0 00 84 0d 88 c8 b1 3d 38 f1
>0f070 24 f0 f8 c9 80 f0 1b b1 24 30 03 c8 d0 f9 c8 e6
>0f080 0d 18 98 65 24 85 24 90 02 e6 25 18 a0 00 b1 24
>0f090 d0 da 05 0d 85 0d 60 a0 00 ca 10 0f b1 24 48 e6
>0f0a0 24 d0 02 e6 25 68 10 f4 30 ef c8 b1 24 30 0a 20
>0f0b0 00 17 20 0c 56 24 0f d0 f1 60 85 24 8a f0 02 a9
>0f0c0 03 a8 a6 24 b9 2d f0 85 24 b9 2e f0 85 25 20 97
>0f0d0 f0 4c 1c f0
```

Und damit Sie auch etwas davon haben, ist hier wieder das kommentierte Assemblerlisting:

```
f000   4c da f0   jmp $f0da    Sprung: Funktionsauswertung
f003   4c 32 f0   jmp $f032    Sprung: Tokenumwandlung
f006   4c ba f0   jmp $f0ba    Sprung: Listroutine
f009   4c 00 f1   jmp $f100    Sprung: Befehlsauswertung
f00c   20 03 17   jsr $1703    Sprungmodul JMP-Befehl
f00f   4c 78 4c   jmp $4c78    Rücksprung in Funktionsauswertung
f012   24 0f      bit $0f      in Bank 15
f014   20 03 17   jsr $1703    Sprungmodul JMP-Befehl
f017   4c 21 43   jmp $4321    Rücksprung in Tokenumwandlung
f01a   24 0f      bit $0f      in Bank 15
f01c   20 03 17   jsr $1703    Sprungmodul JMP-Befehl
f01f   4c 2e 51   jmp $512e    Rücksprung in Listroutine
f022   24 0f      bit $0f      in Bank 15
f024   20 03 17   jsr $1703    Sprungmodul JMP-Befehl
f027   4c a9 4b   jmp $4ba9    Rücksprung in Befehlsauswertung
f02a   24 0f      bit $0f      in Bank 15

f02c   2c 00 f8   bit $f800    Adresse der Befehlsnamen
f02f   2c 00 f9   bit $f900    Adresse der Funktionsnamen
f032   ac 2d f0   ldy $f02d    Adresse der Befehlsnamen
f035   ad 2e f0   lda $f02e    laden
f038   20 62 f0   jsr $f062    und Befehl suchen
f03b   90 06      bcc $f043    Wenn nicht gefunden, Skip
f03d   a2 00      ldx #$00     Flag für Sondertoken 1
f03f   a5 0d      lda $0d      Befehlsnummer als Token laden
f041   d0 0f      bne $f052    Unbedingter Sprung
f043   ac 30 f0   ldy $f030    Adresse der Funktionsnamen
f046   ad 31 f0   lda $f031    laden
f049   20 62 f0   jsr $f062    und Funktion suchen
```

```
f04c  90 08     bcc $f056     Wenn nicht gefunden, Skip
f04e  a2 ff     ldx #$ff      Flag für Sondertoken 2
f050  a5 0d     lda $0d       Funktionsnummer als Token laden
f052  18        clc           Flag für Kommando erkannt setzen
f053  4c 14 f0  jmp $f014     und Token eintragen
f056  20 00 17  jsr $1700     Sprungmodul JSR-Befehl
f059  20 86 03  jsr $0386     Aktuelles Zeichen holen
f05c  24 00     bit $00       Sprung in Bank 0
f05e  38        sec           Flag für Kommando nicht erkannt
f05f  4c 14 f0  jmp $f014     Auswertung fortsetzen

f062  85 25     sta $25       Diese Routine ist zu der
f064  84 24     sty $24       im ROM-Listing ab $43E2
f066  a0 00     ldy #$00      identisch
f068  84 0d     sty $0d
f06a  88        dey           Sie liegt nur deshalb im
f06b  c8        iny           RAM, damit die Texttabellen
f06c  b1 3d     lda ($3d),y   ebenfalls im RAM liegen
f06e  38        sec           können
f06f  f1 24     sbc ($24),y
f071  f0 f8     beq $f06b
f073  c9 80     cmp #$80
f075  f0 1b     beq $f092
f077  b1 24     lda ($24),y
f079  30 03     bmi $f07e
f07b  c8        iny
f07c  d0 f9     bne $f077
f07e  c8iny
f07f  e6 0d     inc $0d
f081  18        clc
f082  98        tya
f083  65 24     adc $24
f085  85 24     sta $24
f087  90 02     bcc $f08b
f089  e6 25     inc $25
f08b  18        clc
f08c  a0 00     ldy #$00
f08e  b1 24     lda ($24),y
f090  d0 da     bne $f06c
f092  05 0d     ora $0d
f094  85 0d     sta $0d
f096  60        rts
```

```
f097   a0 00      ldy #$00       Diese Routine ist im
f099   ca         dex            ROM-Listing ab $516E
f09a   10 0f      bpl $f0ab      fast identisch vorhanden
f09c   b1 24      lda ($24),y
f09e   48         pha            Sie wurde ins RAM verlegt,
f09f   e6 24      inc $24        damit auch die Befehlstexte
f0a1   d0 02      bne $f0a5      im RAM liegen können
f0a3   e6 25      inc $25
f0a5   68         pla
f0a6   10 f4      bpl $f09c
f0a8   30 ef      bmi $f099
f0aa   c8         iny
f0ab   b1 24      lda ($24),y
f0ad   30 0a      bmi $f0b9
f0af   20 00 17   jsr $1700      Sprungmodul JSR-Befehl
f0b2   20 0c 56   jsr $560c      Zeichenausgabe
f0b5   24 0f      bit $0f        in Bank 15
f0b7   d0 f1      bne $f0aa
f0b9   60         rts

f0ba   85 24      sta $24        Tokennummer setzen
f0bc   8a         txa            Sondertoken 1 ?
f0bd   f0 02      beq $f0c1      Ja, Offset gleich 0
f0bf   a9 03      lda #$03       Sonst Offset gleich 3
f0c1   a8         tay
f0c2   a6 24      ldx $24        Tokennummer wieder laden
f0c4   b9 2d f0   lda $f02d,y    Zeiger je nach Offset
f0c7   85 24      sta $24        auf Befehlsnamen oder
f0c9   b9 2e f0   lda $f02e,y    Funktionsnamen setzen
f0cc   85 25      sta $25
f0ce   20 97 f0   jsr $f097      Gesuchtes Kommando listen
f0d1   4c 1c f0   jmp $f01c      Und Rücksprung in LIST
```

Am Anfang des Sprungmoduls sind nun auch alle Sprungvektoren vorhanden, die wir benötigen. Wir haben jetzt schon eine komplette Ergänzung zur Tokenumwandlung und zum LIST-Befehl. Jetzt brauchen wir nur noch Befehls- und Funktionsnamen einzutragen, die Escape-Vektoren für die Tokenumwandlung und den LIST-Befehl zu verbiegen und wir können unsere neuen Funktionen schon eingeben und wieder auflisten. Ab $F02C haben wir die Adressen für die Texttabellen der

Befehle und Funktionen gespeichert. Geben Sie nun bei der ersten Adresse $F800 mit dem Monitor folgende Bytes ein:

```
>0f800 44 4f 4b c5 00
```

Sobald Sie RETURN drücken, sehen Sie, daß unser neuer Befehl *DOKE* heißen wird. Und als Namen für die neue Funktion geben Sie *DEEK* ein:

```
>0f900 44 45 45 cb 00
```

Die Ablage der Befehlstexte ist mit der Ablage der originalen Befehle im ROM-Listing ab $4417 identisch. Bei dem letzten Zeichen jedes Befehlsnamens ist das Bit 7 gesetzt. Jede Tabelle wird mit einem Null-Byte beendet.

Sie sollten jetzt alles erst einmal abspeichern. Nun ändern Sie den Escape-Vektor für die Tokenumwandlung bei ($030C) auf unsere Sprungtabelle in $1788 und den Escape-Vektor des LIST-Befehls bei ($030E) auf $1790. Rufen Sie jetzt aus dem Monitor wieder das BASIC 7.0 auf, und tippen Sie eine kurze Programmzeile mit den neuen Kommandos DOKE und DEEK ein. Sie können mit dem Monitor überprüfen, ob in der Programmzeile ab $1C00 die neuen Kommandos wirklich als Sondertoken abgelegt sind.

Jetzt haben wir zwei neue Befehle, die der Interpreter auch richtig übernimmt. Nur ausführen kann er sie noch nicht. Deshalb folgt jetzt die zweite Hälfte des Befehlsmoduls, die die Verarbeitung neuer Befehle und Funktionen übernimmt:

```
>0f0d4 2c 00 fa 2c 00 fb c9 80 b0 04 38 4c 0c f0 0a a8
>0f0e4 ad d8 f0 85 57 ad d9 f0 85 58 b1 57 48 c8 b1 57
>0f0f4 85 58 68 85 57 20 56 00 18 4c 0c f0 c9 80 90 17
>0f104 0a a8 ad d5 f0 85 57 ad d6 f0 85 58 20 1f f1 20
>0f114 03 17 4c a0 17 24 0f 38 4c 24 f0 b1 57 48 c8 b1
>0f124 57 85 58 68 85 57 20 00 17 20 80 03 24 0f 4c 56
>0f134 00
```

Natürlich wird der Speicherauszug auch wieder dokumentiert:

```
f0d4   2c 00 fa  bit $fa00      Adresse der Befehlsadresstabelle
f0d7   2c 00 fb  bit $fb00      Adresse der Funktionsadresstabelle
f0da   c9 80     cmp #$80       Eine neue Funktion ?
f0dc   b0 04     bcs $f0e2      Ja, Skip
f0de   38        sec            Flag für Funktion nicht erkannt
f0df   4c 24 f0  jmp $f00c      Fehler ausgeben
f0e2   0a        asl            Funktionsnummer * 2
f0e3   a8        tay            ergibt Offset auf Tabelle
f0e4   ad d8 f0  lda $f0d8      Zeiger auf Adreßtabelle
f0e7   85 57     sta $57        der Funktionen setzen
f0e9   ad d9 f0  lda $f0d9
f0ec   85 58     sta $58
f0ee   b1 57     lda ($57),y    Adresse Low laden
f0f0   48        pha            und retten
f0f1   c8        iny
f0f2   b1 57     lda ($57),y    Adresse High laden
f0f4   85 58     sta $58        und setzen
f0f6   68        pla            Adresse Low laden
f0f7   85 57     sta $57        und setzen
f0f9   20 56 00  jsr $0056      Neue Funktion aufrufen
f0fc   18        clc                  Flag für Funktion gefunden
f0fd   4c 0c f0  jmp $f00c      Rücksprung in Ausdruckauswertung

f100   c9 80     cmp #$80       Ein neuer Befehl ?
f102   90 17     bcc $f11b      Nein, Fehler
f104   0a        asl            Befehlsnummer * 2
f105   a8        tay            ergibt Offset auf Tabelle
f106   ad d5 f0  lda $f0d5      Zeiger auf Adreßtabelle
f109   85 57     sta $57        der Befehle setzen
f10b   ad d6 f0  lda $f0d6
f10e   85 58     sta $58
f110   20 1d f1  jsr $f11f      Befehl aufrufen
f113   20 03 17  jsr $1703      Sprungmodul JMP-Befehl
f116   4c a0 17  jmp $17a0      Über RTS Rücksprung ins ROM
f119   24 0f     bit $0fnach Bank 15
f11b   38        sec            Flag für Befehl nicht erkannt
f11c   4c 24 f0  jmp $f024      Fehler ausgeben
f11f   b1 57     lda ($57),y    Adresse Low laden
f121   48        pha            und retten
f122   c8        iny
f123   b1 57     lda ($57),y    Adresse High laden
f125   85 58     sta $58        und setzen
```

```
f127   68         pla         Adresse Low laden
f128   85 57      sta $57     und setzen
f12a   20 00 17   jsr $1700   Sprungmodul JSR-Befehl
f12d   20 80 03   jsr $0380   Aktuelles Zeichen aus Programm
f130   24 0f      bit $0f     holen
f132   4c 56 00   jmp $0056   und Befehl aufrufen
```

Wie auch im ersten Teil des Befehlsmoduls sind die Tabellenadressen wieder in zwei BIT-Befehlen gespeichert, sie lassen sich dann leichter finden und ändern. Zu den Adreßtabellen sei noch etwas gesagt: Es sind immer die absoluten Adressen der Routinen für die Befehle und Funktionen. Das ist speziell bei den Befehlen so. Das anzumerken ist deshalb wichtig, weil der Interpreter seine Befehle etwas anders aufruft und keine absoluten sondern Adreßwerte minus eins benutzt.

Das Befehlsmodul ist nun vollständig und funktionsfähig. Wir brauchen aber noch die Assemblerprogramme für die neuen Kommandos, damit das Befehlsmodul auch etwas zu tun hat. Nehmen wir zuerst den Speicherauszug für den DOKE-Befehl:

```
>0f200 20 00 17 20 12 88 24 0f a5 16 48 a5 17 48 20 00
>0f210 17 20 5c 79 24 0f 20 00 17 20 12 88 24 0f a6 16
>0f220 a4 17 68 85 17 68 85 16 98 48 20 00 17 20 e8 80
>0f230 24 0f e6 16 d0 02 e6 17 68 20 00 17 20 e9 80 24
>0f240 0f 60
```

Zum DOKE-Befehl gibt es natürlich auch ein kommentiertes Assemblerlisting:

```
f200   20 00 17   jsr $1700   Sprungmodul JSR-Befehl
f203   20 12 88   jsr $8812   Adresse auswerten
f206   24 0f      bit $0f     Bank 15
f208   a5 16      lda $16     Adresse retten
f20a   48         pha
f20b   a5 17      lda $17
f20d   48         pha
f20e   20 00 17   jsr $1700   Sprungmodul JSR-Befehl
f211   20 5c 79   jsr $795c   Test auf Komma
f214   24 0f      bit $0f     Bank 15
f216   20 00 17   jsr $1700   Sprungmodul JSR-Befehl
f219   20 12 88   jsr $8812   Adreßwert holen
```

```
f21c   24 0f      bit $0f     Bank 15
f21e   a6 16      ldx $16     Adreßwert in Register retten
f220   a4 17      ldy $17
f222   68         pla         Speicheradresse wieder setzen
f223   85 17      sta $17
f225   68         pla
f226   85 16      sta $16
f228   98         tya         High-Byte des Adreßwertes
f229   48         pha         retten
f22a   20 00 17   jsr $1700   Sprungmodul JSR-Befehl
f22d   20 e8 80   jsr $80e8   (X) Low-Byte ins RAM bringen
f230   24 0f      bit $0f     Bank 15
f232   e6 16      inc $16     Speicheradresse erhöhen
f234   d0 02      bne $f238
f236   e6 17      inc $17
f238   68         pla         High-Byte wieder holen
f239   20 00 17   jsr $1700   Sprungmodul JSR-Befehl
f23c   20 e9 80   jsr $80e9   (A) High-Byte ins RAM bringen
f23f   24 0f      bit $0f     Bank 15
f241   60         rts
```

Wie Sie aus dem Assemblerlisting ersehen können, bringt der DOKE-Befehl einen Adreßwert in zwei aufeinanderfolgende Speicherzellen. Sie müssen einen Adreßwert also nicht mehr mühselig in Low-Byte und High-Byte auftrennen, bevor Sie ihn ins RAM schreiben können. Der DOKE-Befehl - sein Name kommt von Doppel-POKE - berücksichtigt wie der normale POKE-Befehl die Speicherkonfiguration, die Sie mit dem BANK-Befehl anwählen.

Bevor wir den DOKE-Befehl aber endgültig einbinden, kommt die versprochene DEEK-Funktion. Hier ist der entsprechende Speicherauszug:

```
>0f300 20 00 17 20 56 79 24 0f a5 16 48 a5 17 48 20 00
>0f310 17 20 da 77 24 0f 20 00 17 20 15 88 24 0f a0 00
>0f320 a9 16 20 00 17 20 74 ff 24 0f 85 65 e6 16 d0 02
>0f330 e6 17 a0 00 a9 16 20 00 17 20 74 ff 24 0f 85 64
>0f340 68 85 17 68 85 16 a2 90 38 20 00 17 20 75 8c 24
>0f350 0f 60
```

Auch hier wieder das Assemblerlisting dazu:

```
f300   20 00 17  jsr $1700    Sprungmodul JSR-Befehl
f303   20 56 79  jsr $7956    Test auf ')'
f306   24 0f     bit $0f      Bank 15
f308   a5 16     lda $16      Adressspeicher retten
f30a   48        pha
f30b   a5 17     lda $17
f30d   48        pha
f30e   20 00 17  jsr $1700    Sprungmodul JSR-Befehl
f311   20 da 77  jsr $77da    Auf numerisch prüfen
f314   24 0f     bit $0f      Bank 15
f316   20 00 17  jsr $1700    Sprungmodul JSR-Befehl
f319   20 15 88  jsr $8815    FAC#1 in Adressformat wandeln
f31c   24 0f     bit $0f      Bank 15
f31e   a0 00     ldy #$00     Offset auf 0
f320   a9 16     lda #$16     Adresse ist in ($16)
f322   20 00 17  jsr $1700    Sprungmodul JSR-Befehl
f325   20 74 ff  jsr $ff74    Low-Byte aus Speicher holen
f328   24 0f     bit $0f      Bank 15
f32a   85 65     sta $65      Low-Byte setzen
f32c   e6 16     inc $16      Ladeadresse erhöhen
f32e   d0 02     bne $f332
f330   e6 17     inc $17
f332   a0 00     ldy #$00     Offset auf 0
f334   a9 16     lda #$16     Adresse ist in ($16)
f336   20 00 17  jsr $1700    Sprungmodul JSR-Befehl
f339   20 74 ff  jsr $ff74    High-Byte aus Speicher holen
f33c   24 0f     bit $0f      Bank 15
f33e   85 64     sta $64      High-Byte setzen
f340   68        pla          Adressenspeicher wieder setzen
f341   85 17     sta $17
f343   68        pla
f344   85 16     sta $16
f346   a2 90     ldx #$90     Exponent für Umwandlung in
f348   38        sec          Realzahl setzen
f349   20 00 17  jsr $1700    Sprungmodul JSR-Befehl
f34c   20 75 8c  jsr $8c75    Adreßwert in FAC#1
f34f   24 0f     bit $0f      Bank 15
f351   60        rts
```

Die DEEK-Funktion ist das Gegenstück zum DOKE-Befehl. Sie liest einen Adreßwert aus zwei aufeinanderfolgenden Speicherzellen aus. Auch sie berücksichtigt die Speicherkonfiguration, die durch den BANK-Befehl gewählt worden ist.

Das Befehlsmodul ist im Speicher, die Assemblerprogramme und die Namen für die neuen Kommandos ebenfalls, wir müssen nur noch in die Adreßtabellen die Startadressen der Kommandos eintragen. Zuerst für den DOKE-Befehl:

```
>0fa00 00 f2
```

Und dann für die DEEK-Funktion:

```
>fb00 00 f3
```

Sie haben jetzt sehr viel eingetippt. Speichern Sie deshalb alles sicherheitshalber ab. Jetzt hat das Befehlsmodul auch alle Angaben bekommen, die es braucht, um die neuen Kommandos aufzurufen. Setzen Sie also den Escape-Vektor für Funktionsaufrufe bei ($02FC) in die Sprungtabelle auf $1780 und den Escape-Vektor für Befehlsauswertung bei ($0310) auf $1798. Jetzt können Sie den DOKE-Befehl und die DEEK-Funktion auch benutzen. Tippen Sie doch folgendes ein:

```
PRINT HEX$(DEEK(45))
```

Die Startadresse für BASIC-Programme wird nun hexadezimal ausgegeben. Zwei PEEK-Befehle würden hier mehr Arbeit machen. Eine echte Erleichterung bietet aber erst der DOKE-Befehl:

```
DOKE 808, DEC("F670")
```

Diese Eingabe sperrt die STOP-Taste. Rückgängig gemacht werden kann sie durch

```
DOKE 808,DEC("F66E")
```

Hier ändert sich zwar nur das Low-Byte des Adreßwertes, aber dasselbe mit zwei POKE-Befehlen zu erledigen ist nicht ganz so

einfach und erfordert noch einige mathematische Operationen.

Sie können mit dem Befehlsmodul bis zu 128 neue Befehle und bis zu 128 neue Funktionen einführen. Probleme wird es nur mit freiem Speicherplatz geben. Da immer zuerst versucht wird, die neuen Kommandos zu erkennen, ist es auch möglich, alte Befehle durch neue zu ersetzen. Man braucht nur eine neue Funktion mit dem Namen "SQR" zu entwickeln, damit die originale Wurzelfunktion ignoriert wird. Natürlich ändert sich dabei auch das entsprechende Token für die Funktion. Sie sollten deshalb keine Sprungbefehle ersetzen, da sonst zum Beispiel der RENUMBER-Befehl nicht mehr funktioniert. Ansonsten sind Ihrem Ideenreichtum aber keine Grenzen gesetzt.

10. Das Z-80-ROM-Listing

10.1 Einführung zur Z-80

Der Teil des ROM, der beinahe ausschließlich Z-80-Mnemocode beinhaltet, umfaßt alles in allem 4 KByte, der physikalisch im Adreßbereich $D000 bis $DFFF liegt; Kenner spitzen hier schon die Ohren, denn sie wissen, daß es sich hier um einen recht kritischen Speicherbereich handelt.

Ist die Z-80 eingeschaltet (ein Bit im MODE CONFIGURATION REGISTER ist hierfür verantwortlich), so adressiert die MMU, das ist der für die Adressierung zuständige Baustein, das ab $D000 liegende ROM, wenn die Z-80 den Bereich $0000 bis $0FFF adressiert; hier liegt normalerweise bei einem Z-80-Betriebssystem das ROM. Als Anwender oder Programmierer unter Z-80 merken Sie hiervon allerdings garnichts, da die MMU dies alles vollkommen automatisch regelt.

Wird der Rechner eingeschaltet oder ein RESET ausgeführt, so wird die Z-80 auf jeden Fall zunächst einmal kurz eingeschaltet. Es werden die notwendigen Vorbereitungen für einen eventuellen CP/M-Start getroffen; Sie wissen ja, daß der Commodore 128 CP/M automatisch bootet und startet, sollte sich eine CP/M-Diskette beim RESET im Laufwerk befinden. Ist das Z-80-Betriebssystem mit seinen Vorbereitungen fertig, so wird wieder die 8502 eingeschaltet. Diese macht dann genau an der Stelle weiter, an der sie ihre Arbeit beendet hat - dies ist normalerweise die Stelle, nach der die Z-80 eingeschaltet worden ist. Klingt alles zunächst ein wenig kompliziert, ist es eigentlich aber gar nicht.

Es ist sicherlich sehr sinnvoll, das Z-80-Betriebssystem dokumentiert abzudrucken. Wenn Sie unter der Z-80 programmieren wollen, so ist es sicherlich sinnvoll zu wissen, wie das Z-80-ROM aussieht; schließlich müssen Sie damit ja kooperieren. Ferner ist aus dem Programmierstil leicht ersichtlich, wie man die Z-80 beim Commodore 128 programmieren muß, denn es

gibt einige Eigenheiten, die auf das Adreßchaos zurückzuführen sind.

Will man beispielsweise den Bereich $D000 bis $DFFF adressieren, also einen der Bausteine wie VIC, VDC, SID oder einen der anderen, so können Sie dies nicht mit den "normalen" Adressierungskommandos tun. Hierzu sind die Port-Kommandos *OUT* und *IN* zu verwenden. Um beispielsweise die 8502 einzuschalten, ist folgender Befehlsablauf notwendig:

```
LD  BC,$D505   ;Adresse Mode Configuration Register
LD  A,$B1      ;Code, um 8502 einzuschalten
OUT (C),A      ;entspricht LD (BC),A
```

entsprechend müssen Sie zum Auslesen von Speicherstellen das IN-Kommando verwenden. Im Registerpaar BC muß immer die anzusprechende Adresse enthalten sein.

Wenn Sie das hier abgedruckte ROM-Listing sorgsam studieren, so werden Sie feststellen, daß auch einige 8502-Code-Sequenzen enthalten sind. Das Umschalten der Prozessoren beispiesweise muß es ja in beiden Mnemocodes geben, sonst klappt es nicht. Ferner muß sich das Umschalten unbedingt innerhalb des Common Area abspielen, sonst meldet sich der Rechner nach dem Einschalten der Z-80 mit an Sicherheit grenzender Wahrscheinlichkeit nicht mehr wieder.

Wozu aber dient nun das Z-80-ROM, werden Sie sich berechtigterweise fragen. Nun, ganz einfach. Es dient in erster Linie dem Laden und Starten der notwendigen CP/M-Dateien. Ferner befinden sich sämtliche Bildschirmroutinen in diesem ROM, sowohl für den 40- als auch für den 80-Zeichen-Bildschirm. Hierauf greift das BIOS und das BDOS sicherlich zurück, da nicht alle Routinen tatsächlich innerhalb des ROMs genutzt werden. Alle Systemmeldungen, die Sie während des Bootens von CP/M auf dem Bildschirm erhalten, erfolgen aus diesem ROM. Verfallen Sie aber bitte nicht dem Irrtum, daß das gesamte CP/M oder auch nur ganze Teile von CP/M sich in diesem ROM befinden; es handelt sich hierbei lediglich um

extrem systemspezifische Routinen.

Das Laden von Diskette allerdings erfolgt in guter alter 8502-Manier. Hierfür schaltet die Z-80 kontrolliert kurz die 8502 ein, welche nach getaner Arbeit wieder brav zur Z-80 umschaltet. Dies ist eigentlich recht sinnvoll, da man auf diese Weise auf die bestehenden Kernal-Routinen zurückgreifen kann.

Wir wollen Sie nun aber - bevor Sie sich des Z-80-ROMs näher annehmen - kurz aufklären, denn sonst wachsen Ihnen auch noch graue Haare, und das muß ja nicht sein. Sie müssen wissen, daß beim Einschalten der Z-80 nichts mehr so bleibt, wie es ist. Der Bereich $D000 bis $DFFF wird nach $0000 heruntergespiegelt; ein Adressieren des Bereiches ab $D000 ist ausschließlich mittels OUT (C),x und IN (C),x möglich. Aber auch der Bildschirm und der Zeichengenerator werden verschoben; dies ist ganz logisch, denn dort, wo der Videoram sich normalerweise befindet, wird ja ROM adressiert: $0400 bis $07FF können Sie dem ROM-Listing entnehmen, der VIC-Chip wäre recht verwirrt.

Das Videoram wird nach $2C00 verschoben; der Zeichengenerator wird um 256 Byte nach oben verschoben, beginnt also bei $D100. Einzig das Farbram bleibt dort, wo es ist, es läßt sich bekanntlich ja nicht verschieben.

Damit das Ganze aber nicht so einfach ist, gibt es neben dem 40-Zeichen- und dem 80-Zeichen-Bildschirm noch einen dritten, dem 40-Zeichen-Bildschirm übergeordneten Bildschirm. Beim 40-Zeichen-Bildschirm wird von der Anzeige her ein 80-Zeichen-Bildschirm simuliert, der in der Horizontalen gescrollt werden kann. Um dies zu ermöglichen, muß man natürlich einen zusätzlichen Speicher haben. Wir haben diesen Bildschirm 80-Zeichen-Simulator getauft; es wird immer nur ein Fenster aus diesem Bildschirm auf dem 40-Zeichen-Bildschirm dargestellt.

Dieser 80-Zeichen-Bildschirm liegt im Bereich ab $1400; das dazugehörige Farbram finden Sie ab $1C00. Allerdings kann man vom Gebrauch dieses Pseudo-80-Zeichen-Bildschirmes nur abraten, den Überblick dürfte wohl jeder früher oder später verlieren.

So, das wäre eigentlich schon das Wichtigste zum ROM; nun aber noch eine kleine Auflistung der verwendeten Systemadressen im RAM, damit Sie nicht so lange rumrätseln müssen.

$2400	Zwischenspeicher allerlei
$2402	angepaßte Spalte für 80-Zeichen-Simulation
$2404+	Maske für Textausgabe
$2406+	Adresse Cursor 40-Zeichen-Schirm
$2408	Zeilen pro Bildschirm (24 Default)
$2409+	Cursoradresse + $1400 für 80-Zeichen-Simulation
$240B	Cursorspalte (40-Zeichen)
$240C	Cursorzeile (40-Zeichen)
$240D	Zeichenfarbe
$240E	Hintergrundfarbe
$240F	Rahmenfarbe
$2410	Füllzeichen (entweder $00 oder $80 für reverses Space)
$2411+	Cursoradresse
$2413	Cursorspalte
$2414	Cursorzeile
$2415	Attribut
$2416	Hintergrundfarbe 80-Zeichen-Schirm
$2417	Vordergrundfarbe 80-Zeichen-Schirm
$FD01	Flag für Vektoren setzen Ja/Nein
$FD03	zu lesende Spur
$FD04	zu lesender Sektor
$FD05	noch zu ladende Anzahl Blöcke
$FD06	Fehlerflag ($00,$0D,$FF)
$FD08	Logische Filenummer
$FD0D	Zeiger auf Anpassungstabelle
$FD10	Attack/Decay
$FD11	Volume
$FD12	Frequenz (Hi)
$FD13	Sustain/Release
$FD14	Ausschalten Ton
$FD15	Einschalten Ton
$FD18+	Basisadresse zu ladender Block

Mit "+" gekennzeichnete Adressen sind als 16-Bit-Werte zu verstehen.

Der Bootsektor der CP/M-Diskette soll auch noch kurz angeschnitten werden, damit Sie ihn sich nicht selbst anzusehen brauchen. Natürlich beginnt er mit der Identifizierung als Bootsektor, beinhaltet also zunächst die drei Buchstaben "CBM"

und dann fünf Nullen. Das dann zu startende Programm sieht so aus:

```
SEI           Interrupt unterbinden
JSR $FF84     IOINIT
LDA #$3E      Konfigurationsbyte
STA $FF00     definieren
LDA #$C3      Code für JP unter Z-80
STA $FFEE     Code speichern
LDA #$08      Lo-Byte ist $08
STA $FFEF     speichern
LDA #$00      Hi-Byte ist $00
STA $FFF0     speichern; entspricht einem RST $08
JMP $FFD0     Z-80 einschalten
```

Das ist alles, was der Booting-Sektor macht. Alles andere spielt sich innerhalb des Z-80-ROMs ab und ist Ihnen ab sofort nicht mehr verborgen. Es ist Wert darauf gelegt worden, daß nahezu jede Zeile sinnvoll dokumentiert wurde.

Speicherbereiche, die kopiert werden, sind mit zwei Adressen versehen. Zunächst die physikalisch/logische Adresse und dann die vorgesehene Zieladresse. Relative Sprünge beziehen sich auf ihre vorgesehene Speicherumgebung.

Nun aber das versprochene ROM-Listing:

10.2 Das Z-80-ROM-Listing

```
************************************ RST 00 (Kaltstart)

0000:  3E 3E       LD   A,$3E       Konfigurationsbyte (RAM,I/O)
0002:  32 00 FF    LD   ($FF00),A   ins Konfigurationsregister
0005:  C3 3B 00    JP   $003B       Rest des Kaltstarts

************************************ RST 08; Z-80 wieder eingeschaltet

0008:  31 77 3C    LD   SP,$3C77    SP initialisieren
000B:  3E 3F       LD   A,$3F       Konfigurationsbyte
000D:  C3 8C 01    JP   $018C       Rest der RST-08-Routine

************************************ RST 10

0010:  E1          POP  HL          Rücksprungadresse von Stack
0011:  6E          LD   L,(HL)      Folgebyte als Offset holen
0012:  C3 20 00    JP   $0020       RST-20-Routine anspringen

0015:  00          NOP              Füllbytes
0016:  00          NOP
0017:  00          NOP

************************************ RST 18

0018:  E1          POP  HL          Rücksprungadresse von Stack
0019:  6E          LD   L,(HL)      Lo-Byte der Rücksprungadresse
001A:  C3 28 00    JP   $0028       RST-28-Routine anspringen

001D:  00          NOP              Füllbytes
001E:  00          NOP
001F:  00          NOP

************************************ RST 20

0020:  3A 0F FD    LD   A,($FD0F)
0023:  A7          AND  A           Setze Flags
0024:  28 02       JR   Z,$0028     Bei Nullflag Sprung
0026:  2C          INC  L           sonst erhöhe den
0027:  2C          INC  L           Sprungzeiger um 2

************************************ RST 28;
                                    Sprungadresse aus Tabelle holen

0028:  26 01       LD   H,$01       Hi-Byte der Tabelle ist 1
002A:  7E          LD   A,(HL)      Folgebyte als Offset holen
002B:  23          INC  HL          Zeiger nun auf Hi-Byte
```

```
002C:  66              LD    H,(HL)         Hi-Byte auch holen
002D:  6F              LD    L,A            Lo-Byte nach L
002E:  E9              JP    (HL)           Und indirekter Sprung

002F:  00              NOP                  Füllbyte

************************************ RST 30; Dummy; Erstellungsdatum

0030:  30 35 2F 31 32 2F 38 35              .ASC "05/12/85"
                                            =12. Juni 1985

************************************ RST 38

0038:  C3 FD FD        JP    $FDFD          RST 38 bei $FDFD fortführen

************************************ RST 0 Contn'd

003B:  01 2F D0        LD    BC,$D02F       Register 47 des VIC-Chip (Tast.)
003E:  11 FC FF        LD    DE,$FFFC       $FF in die Tastatur schreiben
0041:  ED 51           OUT   (C),D          Keine Erweiterungstasten
0043:  03              INC   BC             Register 48=Taktregister
0044:  ED 59           OUT   (C),E          auf $FC setzen -> 1 MHz-Modus
0046:  01 05 D5        LD    BC,$D505       Mode-Configuration-Register
0049:  3E B0           LD    A,$B0          /EXROM und /GAME testen
004B:  ED 79           OUT   (C),A          sowie 128er-Modus einschalten
004D:  ED 78           IN    A,(C)          Mode-Configuration-Register
004F:  2F              CPL                  wieder auslesen und negieren
0050:  E6 30           AND   $30            /EXROM oder /GAME gesetzt?
0052:  28 05           JR    Z,$0059 Nein, dann keine Cartridge

************************************ 64er-Modus einschalten und
                                     Kontrolle an Cartridge übergeben

0054:  3E F1           LD    A,$F1          8502 einschalten und 64er-Modus
0056:  ED 79           OUT   (C),A          auswählen
0058:  C7              RST   $00            und Kaltstart ausführen
0059:  01 0F DC        LD    BC,$DC0F       CRB-Register im CIA 1
005C:  3E 08           LD    A,$08          auswählen und dann
005E:  ED 79           OUT   (C),A          Timer B anhalten sowie
0060:  0D              DEC   C              auch den Timer A des
0061:  ED 79           OUT   (C),A          CIA 1 stoppen.
0063:  0E 03           LD    C,$03          DDRB-Datenrichtungsregister
0065:  AF              XOR   A              für Port B: Alle Bits auf Ein-
0066:  ED 79           OUT   (C),A          gabe legen
0068:  0D              DEC   C              Zeiger auf DDRA und hier
0069:  3D              DEC   A              alle Bits auf
006A:  ED 79           OUT   (C),A          Ausgabe legen.
006C:  0D              DEC   C              Durch zweimaliges dekremen-
```

```
006D:  0D          DEC   C            tieren zeigt BC auf Port A
006E:  3E 7F       LD    A,$7F        Port A mit $7F (Siehe auch
0070:  ED 79       OUT   (C),A        Tastaturmatrix) beschreiben
0072:  03          INC   BC           Zeiger auf Port B (Eingabe)
0073:  ED 78       IN    A,(C)        und auslesen
0075:  E6 20       AND   $20          Commodore-Taste ausmaskieren
0077:  01 05 D5    LD    BC,$D505     Zeiger für Mode-Config.-Reg.
007A:  28 D8       JR    Z,$0054 Taste war gedrückt-> 64er-Modus
007C:  21 B4 0F    LD    HL,$0FB4     Es werden nun die Register
007F:  01 0A D5    LD    BC,$D50A     der MMU mit den Werten ab
0082:  16 0B       LD    D,$0B        $0FAA belegt
0084:  7E          LD    A,(HL)       Es ist zu beachten, daß alle
0085:  ED 79       OUT   (C),A        11 Register der MMU
0087:  2B          DEC   HL           von hinten mit den Werten
0088:  0D          DEC   C            ab $0FB4 abwärts beschrieben
0089:  15          DEC   D            werden!
008A:  20 F8       JR    NZ,$0084 Ende der Schleife
008C:  21 1A 0D    LD    HL,$0D1A     Den Bereich von $0D1A
008F:  11 00 11    LD    DE,$1100     nach $1100 kopieren
0092:  01 08 00    LD    BC,$0008     Acht Bytes sind zu
0095:  ED B0       LDIR               kopieren (8502-Code!)
0097:  21 E5 0E    LD    HL,$0EE5     Ebenfalls den Bereich
009A:  11 D0 FF    LD    DE,$FFD0     von $0EE5 an nach Common
009D:  01 1F 00    LD    BC,$001F     Area ab $FFD0 kopieren
00A0:  ED B0       LDIR               31 Bytes sind zu kopieren
00A2:  21 00 11    LD    HL,$1100     $1100 als Sprungvektor
00A5:  22 FA FF    LD    ($FFFA),HL   Sprungvektor in
00A8:  22 FC FF    LD    ($FFFC),HL   alle vier Adressen
00AB:  22 FE FF    LD    ($FFFE),HL   kopieren, unter anderem
00AE:  22 DD FF    LD    ($FFDD),HL   auch an $FFDD (frisch kopiert!)
00B1:  C3 E0 FF    JP    $FFE0        und Sprung in Z-80-Teil

*************************************

00B4:  CD 6D 03    CALL  $036D        Lade Blöcke
00B7:  3A 06 3C    LD    A,($3C06)    Hole Blockanzahl
00BA:  22 18 FD    LD    ($FD18),HL   Zieladresse des Ladeblockes
00BD:  11 ED 00    LD    DE,$00ED
00C0:  F5          PUSH  AF           Rette Blockanzahl
00C1:  CD D3 00    CALL  $00D3        Vergleiche (HL) mit (DE)
00C4:  CC FA 02    CALL  Z,$02FA      Wenn ok, dann Aufruf
00C7:  F1          POP   AF           hole Blockanzahl zurück
00C8:  2A 18 FD    LD    HL,($FD18)   Ladeadresse holen
00CB:  11 20 00    LD    DE,$0020     32 als Offset
00CE:  19          ADD   HL,DE        hinzuaddieren
00CF:  3D          DEC   A            erniedrige Blockzähler
00D0:  20 E8       JR    NZ,$00BA     noch nicht fertig, --> schleifen
00D2:  C9          RET                Ende der Unterroutine
```

```
************************************ Vergleiche (HL) mit (DE)

00D3:  06 0C        LD    B,$0C        12 Bytes sind zu vergleichen
00D5:  EB           EX    DE,HL        Austausch beider Vergl.register
00D6:  1A           LD    A,(DE)       Hole ersten Wert in Akku
00D7:  E6 7F        AND   $7F          und lösche irrelevantes 8. Bit
00D9:  BE           CP    (HL)         vergleich mit (HL) durchführen
00DA:  C0           RET   NZ           und bei Unterschied abbrechen
00DB:  23           INC   HL           sonst beide Adressen
00DC:  13           INC   DE           eins hochzählen
00DD:  10 F7        DJNZ  $00D6        und weiterschleifen
00DF:  3A 06 3C     LD    A,($3C06)    Hole Blockanzahl
00E2:  FE 40        CP    $40          64 Blocks zu laden?
00E4:  1A           LD    A,(DE)       Hole Blockanzahl aus Tabelle
00E5:  20 01        JR    NZ,$00E8     $3C06 ist kleiner als 64
00E7:  1F           RRA                sonst verdoppele Akku
00E8:  32 35 3C     LD    ($3C35),A    und merken
00EB:  AF           XOR   A            Akku und Flags löschen
00EC:  C9           RET                Ende der Routine


************************************ Text

00ED:  00           NOP
00EE:  43 50 4D 2B 20 20 20 20         "CPM+    "
00F6:  53 59 53 00                     "SYS" <Ende>

************************************ Vergleich HL mit DE

00FA:  7C           LD    A,H          Hi-Byte von HL
00FB:  BA           CP    D            vergleiche mit D
00FC:  C0           RET   NZ           und Ende, wenn ungleich
00FD:  7D           LD    A,L          sonst vergleiche auch noch
00FE:  BB           CP    E            die Lo-Bytes von HL und DE
00FF:  C9           RET                und Rückkehr mit Flags

************************************ Sprung-Tabelle für RST 28

0100:  84 06        .Word $0684        64 Einsprungadressen
0102:  6E 09        .Word $096E
0104:  AB 06        .Word $06AB
0106:  BC 09        .Word $09BC
0108:  C2 06        .Word $06C2
010A:  DD 09        .Word $09DD
010C:  D1 06        .Word $06D1
010E:  F1 09        .Word $09F1
0110:  DD 06        .Word $06DD
```

```
0112:   31 0A         .Word $0A31
0114:   E8 06         .Word $06E8
0116:   3C 0A         .Word $0A3C
0118:   F1 06         .Word $06F1
011A:   45 0A         .Word $0A45
011C:   7A 07         .Word $077A
011E:   48 0A         .Word $0A48
0120:   80 07         .Word $0780
0122:   62 0A         .Word $0A62
0124:   91 07         .Word $0791
0126:   8E 0A         .Word $0A8E
0128:   CA 07         .Word $07CA
012A:   BA 0A         .Word $0ABA
012C:   DC 07         .Word $07DC
012E:   DF 0A         .Word $0ADF
0130:   1E 08         .Word $081E
0132:   2D 0B         .Word $0B2D
0134:   1B 07         .Word $071B
0136:   7B 0B         .Word $0B7B
0138:   10 07         .Word $0710
013A:   62 0B         .Word $0B62
013C:   1C 09         .Word $091C
013E:   95 09         .Word $0995
0140:   27 09         .Word $0927
0142:   A2 09         .Word $09A2
0144:   4E 07         .Word $074E
0146:   AE 0B         .Word $0BAE
0148:   EB 00         .Word $00EB
014A:   EB 00         .Word $00EB
014C:   EB 00         .Word $00EB
014E:   EB 00         .Word $00EB
0150:   E3 03         .Word $03E3
0152:   6B 04         .Word $046B
0154:   FA 0C         .Word $0CFA
0156:   EB 00         .Word $00EB
0158:   EB 00         .Word $00EB
015A:   EB 00         .Word $00EB
015C:   EB 00         .Word $00EB
015E:   EB 00         .Word $00EB
0160:   3C 0C         .Word $0C3C
0162:   4A 0C         .Word $0C4A
0164:   CF 0B         .Word $0BCF
0166:   0C 0C         .Word $0C0C
0168:   26 05         .Word $0526
016A:   32 05         .Word $0532
016C:   2C 05         .Word $052C
016E:   EB 00         .Word $00EB
0170:   7F 0C         .Word $0C7F
```

```
0172:  C2 0C        .Word $0CC2
0174:  C7 0C        .Word $0CC7
0176:  E4 0C        .Word $0CE4
0178:  EB 00        .Word $00EB
017A:  3D 08        .Word $083D
017C:  AE 08        .Word $08AE
017E:  60 06        .Word $0660
0780:  C3 0A        .Word $0AC3

0182:  09           ADD  HL,BC       Addiere BC als Offset
0183:  C3 33 09     JP   $0933       Hole A:Attribut, B:Zeichen an
                                     (HL) in VDC

************************************ HL als Updateadresse im VDC

0186:  C3 53 09     JP   $0953       HL als Updateadresse im VDC

************************************ VDC-Status abwarten und Register
                                     <Akku> anwählen

0189:  C3 45 09     JP   $0945       VDC-Status abwarten und <Akku>
                                     anwählen

************************************ RST 8 Contn'd

018C:  32 00 FF     LD   ($FF00),A   Akku ins Konfigurationsbyte
018F:  21 00 30     LD   HL,$3000    Es wird der Bereich
0192:  11 01 30     LD   DE,$3001    $3000 bis $FEFF
0195:  01 FF CE     LD   BC,$CEFF    mit dem Wert $00
0198:  75           LD   (HL),L      gefüllt
0199:  ED B0        LDIR
019B:  21 22 0D     LD   HL,$0D22    Der Bereich ab $0D22
019E:  11 00 30     LD   DE,$3000    wird nach $3000 kopiert
01A1:  01 C3 01     LD   BC,$01C3    Es handelt sich hierbei um
01A4:  ED B0        LDIR             8502-Code!
01A6:  21 E5 0E     LD   HL,$0EE5    Bereich ab
01A9:  11 D0 FF     LD   DE,$FFD0    $0EE5 in Common Area
01AC:  01 1F 00     LD   BC,$001F    ab $FFD0 kopieren
01AF:  ED B0        LDIR             31 Bytes
01B1:  3E C9        LD   A,$C9       Code für RETurn
01B3:  32 EE FF     LD   ($FFEE),A   RST 8 durch RET ersetzen
01B6:  CD E0 FF     CALL $FFE0       8502 einschalten - weitermachen
01B9:  21 B4 0F     LD   HL,$0FB4    Die MMU-Register
01BC:  01 0A D5     LD   BC,$D50A    werden mit der Tabelle
01BF:  16 0B        LD   D,$0B       ab $0FAA gefüllt.
01C1:  7E           LD   A,(HL)      s.o.
01C2:  ED 79        OUT  (C),A       s.o.
01C4:  2B           DEC  HL          s.o.
```

```
01C5:  0D              DEC   C            s.o.
01C6:  15              DEC   D            s.o.
01C7:  20 F8           JR    NZ,$01C1     Ende der Schleife
01C9:  21 00 10        LD    HL,$1000     Den Bereich
01CC:  11 01 10        LD    DE,$1001     $1000 bis $2FFF
01CF:  01 FF 1F        LD    BC,$1FFF     mit dem Wert $00
01D2:  75              LD    (HL),L       füllen
01D3:  ED B0           LDIR               s.o.
01D5:  3E 1A           LD    A,$1A        Register 26 (Farben) des VDC
01D7:  CD 45 09        CALL  $0945        anwählen und dann Vorder/
01DA:  3E 90           LD    A,$90        Hintergrundfarben
01DC:  ED 79           OUT   (C),A        VDC auf $90 (Hellroter Cursor)
01DE:  3E 83           LD    A,$83        Hellblau & Alternate-Zeichensatz
01E0:  32 15 24        LD    ($2415),A    als Attribut definieren (VDC)
01E3:  3E 0E           LD    A,$0E        Zeichenfarbe für VIC-Chip
01E5:  32 0D 24        LD    ($240D),A    definieren
01E8:  CD BD 05        CALL  $05BD        Präpariere VDC-Zeichensatz
01EB:  3E 19           LD    A,$19        Der Bildschirm wird mit 24
01ED:  32 08 24        LD    ($2408),A    Zeilen definiert (DEVICE)
01F0:  CD 26 05        CALL  $0526        Nachfolgenden Text ausgeben
01F3:  FF              .Byte $FF          Bildschirm löschen ($FF)
01F4:  81 0A           .Byte $81,$0A      Zeile 1, Spalte 10

01F5:  42 4F 4F 54 49 4E 47 20            BOOTING
01FD:  43 50 2F 4D 20 50 4C 55            CP/M PL
0205:  53 00                              US<Ende>


0208:  01 18 D0        LD    BC,$D018     Basisadresse von Video-RAM
020B:  3E B6           LD    A,$B6        B->10-13 Video-RAM, 6->11-13
020D:  ED 79           OUT   (C),A        CHARROM; Videoram ab $2C00
020F:  CD D2 02        CALL  $02D2        Bootsektor checken und holen
0212:  C2 FF 04        JP    NZ,$04FF     evtl. Fehler ausgeben
0215:  21 B2 0F        LD    HL,$0FB2     Tabellenanfang
0218:  22 02 3C        LD    ($3C02),HL   in $3C02 merken
021B:  CD B4 00        CALL  $00B4        Ersten Teil laden
021E:  CD B4 00        CALL  $00B4        Zweiten Teil laden
0221:  2A 09 3C        LD    HL,($3C09)   Hole Wert
0224:  7C              LD    A,H          Setzen des Zeroflags
0225:  B5              OR    L            wenn 16-Bit-Wert null ist
0226:  CA FF 04        JP    Z,$04FF      und Sprung, falls ja
0229:  21 09 3C        LD    HL,$3C09     Neuer Tabellenanfang
022C:  22 02 3C        LD    ($3C02),HL   merken
022F:  CD 6D 03        CALL  $036D        Daten laden
0232:  21 00 34        LD    HL,$3400     12 Bytes
0235:  11 29 3C        LD    DE,$3C29     von $3400
0238:  01 0C 00        LD    BC,$000C     nach $3C29
023B:  ED B0           LDIR               kopieren
```

```
023D:  CD 26 05      CALL   $0526          Leerzeile ausgeben
0240:  8A            .Byte  $8A,$00,$00    Zeile 10, Spalte 0 und <Ende>
0243:  21 80 34      LD     HL,$3480       Zeiger auf Ausgabetext
0246:  CD 34 05      CALL   $0534          und Text ab (HL) ausgeben
0249:  21 00 35      LD     HL,$3500       Zeiger $3500
024C:  22 04 3C      LD     ($3C04),HL     merken
024F:  CD 26 05      CALL   $0526          Nachfolgenden Text ausgeben
0252:  83            .Byte  $83,$0C        Zeile 3, Spalte 12

0254:  44 41 54 41 20 54 41 42             DATA TAB
025C:  4C 45 53 00                         LES<Ende>

0260:  2A 33 3C      LD     HL,($3C33)     Nacheinander CP/M-Segmente
0263:  22 09 FD      LD     ($FD09),HL     in den Arbeitsspeicher
0266:  21 32 3C      LD     HL,$3C32       einladen
0269:  CD 31 03      CALL   $0331          Hole Blockzahl und Startadresse
026C:  22 0B FD      LD     ($FD0B),HL     Startadresse merken
026F:  CD 44 03      CALL   $0344
0272:  11 80 00      LD     DE,$0080       Recordoffset (CP/M-Intern)
0275:  19            ADD    HL,DE          von 128 addieren
0276:  20 F7         JR     NZ,$026F       Wenn nicht fertig, dann nochmal
0278:  CD 26 05      CALL   $0526          Nachfolgenden Text ausgeben
027B:  84 0C         .Byte  $84,$0C        Zeile 4, Spalte 12

027D:  43 4F 4D 4D 4F 4E 20 43             COMMON C
0285:  4F 44 45 00                         ODE<Ende>

0289:  21 2A 3C      LD     HL,$3C2A       Basisadresse für Tabelle
028C:  CD 24 03      CALL   $0324          und Common Code einladen
028F:  CD 26 05      CALL   $0526          Nachfolgenden Text ausgeben
0292:  85 0C         .Byte  $85,$0C        Zeile 5, Spalte 12

0294:  42 41 4E 4B 45 44 20 43             BANKED C
029C:  4F 44 45 00                         ODE<Ende>

02A0:  21 2C 3C      LD     HL,$3C2C       Basisadresse für Tabelle
02A3:  CD 24 03      CALL   $0324          und Banked Code einladen
02A6:  CD 26 05      CALL   $0526          Nachfolgenden Text ausgeben
02A9:  86 0C         .Byte  $86,$0C        Zeile 6, Spalte 12

02AB:  42 49 4F 53 38 35 30 32             BIOS8502
02B3:  20 43 4F 44 45 00                   CODE<Ende>

02B9:  21 30 3C      LD     HL,$3C30       Basisadresse für Tabelle
02BC:  CD 24 03      CALL   $0324          und BIOS8502 Daten laden
02BF:  3A 30 3C      LD     A,($3C30)
02C2:  47            LD     B,A
02C3:  3A 2F 3C      LD     A,($3C2F)
```

```
02C6:  90          SUB   B             Differenz bilden
02C7:  32 DE FF    LD    ($FFDE),A     Differenz als Hi-Byte
02CA:  AF          XOR   A             Akku löschen (=0)
02CB:  32 DD FF    LD    ($FFDD),A     als Lo-Byte merken
02CE:  2A 2D 3C    LD    HL,($3C2D)    Sprungadresse für CP/M-Einsprung
02D1:  E9          JP    (HL)          holen und anspringen

************************************ Lies ab Spur 1/Sektor 0

02D2:  21 00 FE    LD    HL,$FE00      Ladeadresse für ersten zu
02D5:  22 18 FD    LD    ($FD18),HL    ladenden Block merken
02D8:  AF          XOR   A             Akku löschen
02D9:  32 04 FD    LD    ($FD04),A     Sektor#=0
02DC:  3C          INC   A             Akku=1
02DD:  32 03 FD    LD    ($FD03),A     Spur definieren
02E0:  CD 4F 04    CALL  $044F         Lies Spur/Sektor
02E3:  CD 6B 04    CALL  $046B         Gelesen? Test auf Bootsektor
02E6:  C0          RET   NZ            Kein Bootsektor
02E7:  3C          INC   A             Letztes Zeichen des Blockes+1
02E8:  21 00 38    LD    HL,$3800      Startadresse
02EB:  3E 20       LD    A,$20         32 als Blockzähler
02ED:  20 03       JR    NZ,$02F2      Wenn Akku vor INC<>$FF, Sprung
02EF:  26 3C       LD    H,$3C         sonst Hi-Byte auf $3C ändern
02F1:  87          ADD   A,A           Blockzahl * 2 (Recordanzahl)
02F2:  22 07 3C    LD    ($3C07),HL    Merke Ladeadresse
02F5:  32 06 3C    LD    ($3C06),A     Merke Blockzahl
02F8:  AF          XOR   A             Akku und Flags (Wichtig) löschen
02F9:  C9          RET                 und Ende der Routine

************************************

02FA:  11 09 3C    LD    DE,$3C09      Zieladresse für Kopie
02FD:  3A 35 3C    LD    A,($3C35)     Flag holen
0300:  B7          OR    A             Setze Flags
0301:  28 07       JR    Z,$030A       Flag ist nicht gesetzt
0303:  11 19 3C    LD    DE,$3C19      sonst Startadresse +16
0306:  3D          DEC   A             und Flag (Zähler) erniedrigen
0307:  C2 80 04    JP    NZ,$0480      und Fehler ausgeben

030A:  2A 18 FD    LD    HL,($FD18)    Hole Zieladresse
030D:  01 10 00    LD    BC,$0010      16 als Inkrement
0310:  09          ADD   HL,BC         addieren
0311:  ED B0       LDIR                und 16 Bytes kopieren
0313:  3A 09 3C    LD    A,($3C09)     Zähler holen und
0316:  B7          OR    A             Flags setzen
0317:  C8          RET   Z             Bei null ist alle Arbeit getan
0318:  2A 18 3C    LD    HL,($3C18)    Sonst hole Adresse
031B:  AF          XOR   A             und vergleiche mit $0000
```

```
031C:  BD          CP    L            Akku ist null
031D:  28 02       JR    Z,$0321      Lo-Byte ist null
031F:  BC          CP    H            vergleiche mit Hi-Byte
0320:  C8          RET   Z            Hi-Byte ist null
0321:  C3 29 02    JP    $0229        Zeroflag als Parameter übergeben

*********************************** Lade Records von Floppy

0324:  CD 31 03    CALL  $0331        Hole Anzahl und Startadresse
0327:  11 80 FF    LD    DE,$FF80     128 als Zweierkomplement
032A:  19          ADD   HL,DE        addieren (abziehen!)
032B:  CD 44 03    CALL  $0344        Record laden
032E:  20 F7       JR    NZ,$0327     noch ein Record
0330:  C9          RET                sonst Ende der Routine

*********************************** Hole Anzahl und Startadresse

0331:  5E          LD    E,(HL)       Anzahl holen
0332:  16 00       LD    D,$00        und Hi-Byte löschen
0334:  7B          LD    A,E          Testet Anzahl
0335:  B7          OR    A            auf null
0336:  CA 1B 05    JP    Z,$051B      BAD wenn null
0339:  EB          EX    DE,HL        sonst Anzahl nach HL
033A:  29          ADD   HL,HL        und verdoppeln (Recordanzahl)
033B:  22 00 3C    LD    ($3C00),HL   in $3C00 merken
033E:  EB          EX    DE,HL        wieder nach DE
033F:  2B          DEC   HL           Tabellenzeiger erniedrigen
0340:  66          LD    H,(HL)       Hi-Byte holen
0341:  2E 00       LD    L,$00        und Lo-Byte löschen
0343:  C9          RET                Ende der Routine

*********************************** Record laden und Zähler erniedrigen

0344:  E5          PUSH  HL           Auf Stack retten
0345:  2A 07 3C    LD    HL,($3C07)   Hole erste Vergleichsadresse
0348:  EB          EX    DE,HL        und in DE merken
0349:  2A 04 3C    LD    HL,($3C04)   Zweite Vergleichsadresse holen
034C:  CD FA 00    CALL  $00FA        Vergleiche (HL) mit (DE)
034F:  CC 6D 03    CALL  Z,$036D      und Sprung, wenn gleich
0352:  EB          EX    DE,HL        sonst Register austauschen
0353:  21 80 00    LD    HL,$0080     Record-Offset von 128
0356:  19          ADD   HL,DE        hinzuaddieren
0357:  22 04 3C    LD    ($3C04),HL   und wieder merken
035A:  E1          POP   HL           Adresse zurückholen
035B:  E5          PUSH  HL           und erneut auf Stack sichern
035C:  EB          EX    DE,HL        Ziel und Quelle vertauschen
035D:  01 80 00    LD    BC,$0080     und 128 Bytes kopieren
0360:  ED B0       LDIR               (1 Record)
```

```
0362:  2A 00 3C     LD    HL,($3C00)   Hole Recordzähler
0365:  2B           DEC   HL           um eins erniedrigen
0366:  22 00 3C     LD    ($3C00),HL   und wieder abspeichern
0369:  7D           LD    A,L          Teste, ob Recordanzahl
036A:  B4           OR    H            gleich null ist und setze Flags
036B:  E1           POP   HL           Adresse zurückholen
036C:  C9           RET                und Routine beenden

*********************************** Daten ab $3400 + Offset laden

036D:  21 00 34     LD    HL,$3400     Ladeadresse (Basis)
0370:  22 18 FD     LD    ($FD18),HL   für 8502-Code merken
0373:  E5           PUSH  HL           und auf Stack sichern
0374:  2A 02 3C     LD    HL,($3C02)   Zeiger auf Blockladetabelle
0377:  16 00        LD    D,$00        Hi-Byte löschen
0379:  5E           LD    E,(HL)       und Lo-Byte aus Tabelle holen
037A:  23           INC   HL           Zeiger auf Tabelle erhöhen
037B:  22 02 3C     LD    ($3C02),HL   und wieder merken
037E:  EB           EX    DE,HL        Anzahl Blocks nach HL
037F:  29           ADD   HL,HL        und verdoppeln -> Recordanzahl
0380:  29           ADD   HL,HL        nochmal verdoppeln
0381:  3A 06 3C     LD    A,($3C06)    Hole Blockanzahl
0384:  0F           RRCA               /2
0385:  0F           RRCA               /4
0386:  0F           RRCA               /8
0387:  FE 04        CP    $04          Ist Blockzahl 32 - 63??
0389:  28 01        JR    Z,$038C      Ja, dann Sprung
038B:  29           ADD   HL,HL        sonst verdopple HL erneut
038C:  22 16 FD     LD    ($FD16),HL   uns als Blocknummer merken
038F:  3D           DEC   A            erniedrige Blockanzahl
0390:  32 05 FD     LD    ($FD05),A    und ebenfalls merken
0393:  F5           PUSH  AF           Rette Anzahl auf Stack
0394:  3E 01        LD    A,$01        Anzahl zu ladenden Datenblöcke
0396:  32 BD 31     LD    ($31BD),A    auf 1 setzen
0399:  CD E3 03     CALL  $03E3        Mache aus Block# -> Track/Sektor
039C:  2A 16 FD     LD    HL,($FD16)   Hole Blocknummer
039F:  23           INC   HL           um eins erhöhen und
03A0:  22 16 FD     LD    ($FD16),HL   wieder ablegen
03A3:  F1           POP   AF           Hole Zähler und Flags
03A4:  28 2A        JR    Z,$03D0      Wenn Ende, dann Sprung
03A6:  3A 08 FD     LD    A,($FD08)    Sonst hole Fehlerflag
03A9:  A7           AND   A            und setze Flags
03AA:  28 24        JR    Z,$03D0      keine Fehler
03AC:  2A 03 FD     LD    HL,($FD03)   Track/Sektor holen
03AF:  E5           PUSH  HL           und auf Stack sichern
03B0:  CD E3 03     CALL  $03E3        Block# in Track/Sector wandeln
03B3:  E1           POP   HL           errechneten Track/Sektor holen
03B4:  3A 03 FD     LD    A,($FD03)    Hole Spur#
```

```
03B7:  BD         CP    L            vergleiche mit errechneter Spur
03B8:  20 13      JR    NZ,$03CD     ist leider ungleich
03BA:  E5         PUSH  HL           Rette Track/Sektor
03BB:  2A 16 FD   LD    HL,($FD16)   Hole Block#
03BE:  23         INC   HL           und erhöhe Blocknummer
03BF:  22 16 FD   LD    ($FD16),HL   Neue Blocknummer merken
03C2:  21 BD 31   LD    HL,$31BD     Zeiger auf zu ladende Blockzahl
03C5:  34         INC   (HL)         und ebenfalls erhöhen
03C6:  21 05 FD   LD    HL,$FD05     noch zu ladende Blockzahl
03C9:  35         DEC   (HL)         um eins erhöhen (Korrektur)
03CA:  20 E4      JR    NZ,$03B0     noch wenigstens ein Block
03CC:  E1         POP   HL           Hole Track/Sektor von Stack
03CD:  22 03 FD   LD    ($FD03),HL   Merke Track/Sektor

*********************************** Fehler aufgetreten

03D0:  CD 4F 04   CALL  $044F        Lies Block/Blöcke von Diskette
03D3:  21 19 FD   LD    HL,$FD19     Hi-Byte Zieladresse
03D6:  3A BD 31   LD    A,($31BD)    Hole Anzahl zu ladender Blöcke
03D9:  86         ADD   A,(HL)       zur Zieladresse hinzuaddieren
03DA:  77         LD    (HL),A       und merken
03DB:  3A 05 FD   LD    A,($FD05)    Hole Anzahl zu ladender Blöcke
03DE:  A7         AND   A            Flags setzen
03DF:  20 AE      JR    NZ,$038F     weitermachen, wenn nicht Ende
03E1:  E1         POP   HL           sonst hole Track/Sektor
03E2:  C9         RET                und Ende der Routine

************************************ Aus Block# Track/Sektor machen

03E3:  3E 23      LD    A,$23        Offset 35 für 1571 (2. Seite)
03E5:  32 00 24   LD    ($2400),A    und Offset merken
03E8:  2A 16 FD   LD    HL,($FD16)   Hole Block#
03EB:  11 A8 02   LD    DE,$02A8     ab Blocknummer $2A8 -> 2.Seite
                                     adressieren
03EE:  B7         OR    A            Carry für Subtraktion löschen
03EF:  ED 52      SBC   HL,DE        abs. Blocknummer auf 2. Seite
03F1:  30 05      JR    NC,$03F8     durch Subtraktion ermitteln.
                                     Sprung bei 1. Seite
03F3:  AF         XOR   A            Offset löschen
03F4:  32 00 24   LD    ($2400),A    und speichern
03F7:  19         ADD   HL,DE        Korrektur zur Subtraktion
03F8:  23         INC   HL           Block# + 2, da 1./2. Block
03F9:  23         INC   HL           für Directory reserviert sind
03FA:  11 65 01   LD    DE,$0165     (357) Blocknummer ab Spur 19
03FD:  01 00 15   LD    BC,$1500     Spur 0- hat 21 Sektoren/Spur
0400:  B7         OR    A            Carry für Subtraktion löschen
0401:  ED 52      SBC   HL,DE        Kontrolle auf 21 Sektoren/Spur
0403:  38 1B      JR    C,$0420      Block im 21-Sektoren-Bereich
```

```
0405:  23        INC   HL          +1 zur Fehlerkorrektur
0406:  11 85 00  LD    DE,$0085    nächsten 133 Blöcke haben
0409:  01 11 13  LD    BC,$1311    17 Sektoren pro Spur
040C:  ED 52     SBC   HL,DE       Teste, ob Block in Bereich
040E:  38 10     JR    C,$0420     Jawohl, dann Ende der Testreihe
0410:  11 6C 00  LD    DE,$006C    Nächsten Spuren haben
0413:  01 18 12  LD    BC,$1218    18 Sektoren/Spur (ab Spur $18)
0416:  ED 52     SBC   HL,DE       Teste, ob Block in Bereich
0418:  38 06     JR    C,$0420     Ja, dann Ende der Testreihe
041A:  11 00 00  LD    DE,$0000    Korrekturfaktor auf null
041D:  01 1E 11  LD    BC,$111E    ab Spur 30 -> 17 Sektoren
0420:  19        ADD   HL,DE       Korrektur der Subtraktion
0421:  16 00     LD    D,$00       Hi-Byte von DE löschen
0423:  58        LD    E,B         Sektor nach E
0424:  B7        OR    A           Carry löschen
0425:  0C        INC   C           Spur erhöhen
0426:  ED 52     SBC   HL,DE       Sektoren pro Spur subtrahieren
0428:  30 FB     JR    NC,$0425    und evtl. weiterschleifen
042A:  19        ADD   HL,DE       Fehlerkorrektur
042B:  3A 00 24  LD    A,($2400)   Hole Offset für Seite 0/1
042E:  81        ADD   A,C         addiere Offset zur Spur
042F:  32 03 FD  LD    ($FD03),A   Merke errechnete Spur
0432:  E5        PUSH  HL          Rette Sektor# auf Stack
0433:  21 B5 0F  LD    HL,$0FB5    Tabelle für angepaßte
                                   Sektornummern
0436:  01 15 00  LD    BC,$0015    zur Zugriffsoptimierung
0439:  7B        LD    A,E         Hole Sektornummer
043A:  B9        CP    C           und vergleiche mit Maximalwert
043B:  28 0A     JR    Z,$0447     Ist 21, dann Ende
043D:  09        ADD   HL,BC       addiere Offs. für Tabellenzeiger
043E:  0B        DEC   BC          Nächster Bereich hat 2 Sektoren
043F:  0B        DEC   BC          weniger pro Spur
0440:  B9        CP    C           Ist Maximalwert nun erreicht?
0441:  28 04     JR    Z,$0447     Ja, dann Ende
0443:  09        ADD   HL,BC       addiere Offs. für Folgebereich
0444:  0B        DEC   BC          Folgebereich hat einen Sektor
0445:  18 F9     JR    $0440       weniger und weiterversuchen
0447:  C1        POP   BC          Hole Sektor# von Stack
0448:  09        ADD   HL,BC       und Basis hinzuaddieren
0449:  7E        LD    A,(HL)      Hole angepaßte Sektornummer
044A:  32 04 FD  LD    ($FD04),A   und merken
044D:  3C        INC   A           Flags löschen und
044E:  C9        RET               Ende der Blockberechnung

************************************ Lesen von Floppy

044F:  3E 03     LD    A,$03       Anzahl der Leseversuch auf
0451:  32 36 3C  LD    ($3C36),A   Floppy setzen
```

```
0454:  3E 01          LD    A,$01          Flag für Vektor setzen aus
0456:  32 01 FD       LD    ($FD01),A      (werden nicht nochmal gesetzt)
0459:  CD 8C 05       CALL  $058C          Block (Track/Sektor) auf
                                           Bildschirm anzeigen
045C:  CD E0 FF       CALL  $FFE0          8502 einschalten
045F:  3E 3F          LD    A,$3F          Konfigurationsbyte RAM Bank 0
0461:  32 00 FF       LD    ($FF00),A      einschalten
0464:  3A 06 FD       LD    A,($FD06)      Hole Lesefehlerflag
0467:  B7             OR    A              und teste es auf Lesefehler
0468:  20 32          JR    NZ,$049C       Lesefehler aufgetreten
046A:  C9             RET                  Sonst Ende der Routine

*********************************** Testet geladenen Block auf
                                    Bootsektor

046B:  21 00 FE       LD    HL,$FE00       Startadresse geladener Block
046E:  7E             LD    A,(HL)         Hole erstes Zeichen
046F:  FE 43          CP    $43            ist es "C"?
0471:  C0             RET   NZ             Nein, dann kein Bootsektor
0472:  2C             INC   L              nächstes Zeichen
0473:  7E             LD    A,(HL)         holen und
0474:  FE 42          CP    $42            auf "B" vergleichen
0476:  C0             RET   NZ             Nicht, dann Ende
0477:  2C             INC   L              drittes Zeichen wird über-
0478:  7E             LD    A,(HL)         prüft,
0479:  FE 4D          CP    $4D            auf "M"
047B:  C0             RET   NZ             Nein, kein Bootsektor
047C:  2E FF          LD    L,$FF          Zeiger auf letztes Zeichen
047E:  7E             LD    A,(HL)         Hole dieses Zeichen
047F:  C9             RET                  und Ende der Routine

*********************************** Fehler aufgetreten

0480:  CD 26 05       CALL  $0526          Nachfolgenden Text ausgeben
0483:  93 05          .Byte $93,$05        Zeile 19, Spalte 5

0485:  33 32 4B 20 4D 41 58 20             32K MA:
048D:  43 50 4D 2B 2E 53 59 53             CPM+.SYS
0495:  20 53 49 5A 45 00                    SIZE<Ende>

049B:  CF             RST   $08            Bootvorgang wiederholen

049C:  3C             INC   A              Test auf Akku=&FF
049D:  28 FC          JR    Z,$049B        Bei &FF ebenfalls Neu-Booting
049F:  3A 36 3C       LD    A,($3C36)      Sonst holde den Lesezähler
04A2:  3D             DEC   A              und um eins erniedrigen
04A3:  32 36 3C       LD    ($3C36),A      wieder ablegen
04A6:  20 AC          JR    NZ,$0454       Nochmal versuchen
```

```
04A8:  CD 26 05        CALL   $0526       Nachfolgenden Text ausgeben
04AB:  93 05           .Byte $93,$05      Zeile 19, Spalte 5

04AD:  52 45 41 44 20 45 52               READ ER
04B4:  52 4F 52 00                        ROR<Ende>

04B8:  CD 26 05        CALL   $0526       Nachfolgenden Text ausgeben

04BB:  20 2D 20 48 49 54 20 52             - HIT R
04C5:  45 54 55 52 4E 20 54 4F            ETURN TO
04CD:  20 52 45 54 52 59                   RETRY

04D1:  94 0F           .Byte $94,$0F      Zeile $14,$0F

04D3:  44 45 4C 20 54 4F 20 45            DEL TO E
04DB:  4E 54 45 52 20 43 31 32            NTER C12
04E3:  38 20 4D 4F 44 45 00               8 MODE<Ende>

04EA:  01 00 DC        LD     BC,$DC00    Port A CIA1
                                          (Tastaturdekodierung)
04ED:  3E FE           LD     A,$FE       DEL- und <CR>-Taste werden
                                          ausmaskiert
04EF:  ED 79           OUT    (C),A       und abgecheckt
04F1:  0C              INC    C           Zeiger auf Port B des CIA1
04F2:  ED 78           IN     A,(C)       Hole Ergebnis
04F4:  E6 02           AND    $02         Bit für <CR> testen
04F6:  28 A3           JR     Z,$049B     <CR> ist gedrückt -> Reboot
04F8:  ED 78           IN     A,(C)       Sonst hole Wert erneut
04FA:  E6 01           AND    $01         Teste DEL-Bit
04FC:  20 EC           JR     NZ,$04EA    Nicht gedrückt, weiterversuchen
04FE:  C7              RST    $00         Sonst in den C-128-Modus

04FF:  CD 26 05        CALL   $0526       Nachfolgenden Text ausgeben
0502:  93 05           .Byte $93,$05      Zeile 19, Spalte 5

0504:  4E 4F 00                           NO<Ende>

0507:  CD 26 05        CALL   $0526       Nachfolgenden Text ausgeben
050A:  20 43           .Byte $20,$43      $20=Ab Cursorposition

050B:  43 50 4D 2B 2E 53 59 53            CPM+.SYS
0513:  20 46 49 4C 45 00                   FILE<Ende>

0519:  18 9D           JR     $04B8       Neuer Versuch oder 128er-Modus
051B:  CD 26 05        CALL   $0526       Nachfolgenden Text ausgeben
```

```
051E:  93           .Byte $93,$05      Zeile $13, Spalte $05

0520:  42 41 44 00                     BAD<Ende>

0524:  18 E1        JR    $0507        Neuer Versuch oder 128er-Modus

************************************ Nachfolgenden Text ausgeben

0526:  E3           EX    (SP),HL      Rücksprungadresse von Stack
0527:  CD 34 05     CALL  $0534        Text ab HL bis $00 ausgeben
052A:  E3           EX    (SP),HL      Ende des Textes als neue Rück-
052B:  C9           RET                sprungadresse und RETurn

************************************ Text ab DE ausgeben

052C:  21 FF FF     LD    HL,$FFFF     zeichenmaske für auszugebende
052F:  22 04 24     LD    ($2404),HL   Zeichen setzen
0532:  D5           PUSH  DE           Textadresse auf Stack
0533:  E1           POP   HL           und in HL einlesen

************************************ Text ab HL ausgeben

0534:  56           LD    D,(HL)       Hole Zeichen
0535:  23           INC   HL           Erhöhe Zeiger auf Textstelle
0536:  3A 05 24     LD    A,($2405)    Maske holen
0539:  A7           AND   A            Flags setzen
053A:  28 05        JR    Z,$0541      Maske erlaubt alles, Ende
053C:  AA           XOR   D            Sonst mit Maske verknüpfen
053D:  32 05 24     LD    ($2405),A    und zurückschreiben
0540:  57           LD    D,A          Zeichen nach D

0541:  7A           LD    A,D          aktuelles Zeichen
0542:  B7           OR    A            Flags setzen
0543:  C8           RET   Z            Null ist Endekennzeichen
0544:  FE 24        CP    $24          Dollarzeichen?
0546:  C8           RET   Z            Wenn ja, dann Ende
0547:  E5           PUSH  HL           Rette den aktuellen Zeiger
0548:  21 33 05     LD    HL,$0533     Sprung von Adresse $0533
054B:  E5           PUSH  HL           simulieren
054C:  FE 0A        CP    $0A          Wenn Linefeed, dann
054E:  C8           RET   Z            Sprung an diese Adresse
054F:  FE 0D        CP    $0D          Ist Zeichen <CR>?
0551:  20 0B        JR    NZ,$055E     Nein, dann nach $055E
0553:  CD 45 0A     CALL  $0A45        Spalte=0 - 40 Zeichen
0556:  CD F1 06     CALL  $06F1        Spalte=0 - 80 Zeichen
0559:  CD F1 09     CALL  $09F1        Zeilenzeiger erhöhen
055C:  DF           RST   $18          -> $06D1 (Zeilenzeiger erhöhen)
055D:  0C           .Byte $0C          Sprungvektor
```

```
055E:  FE FF      CP    $FF        Ist Zeichen $FF?
0560:  20 17      JR    NZ,$0579   Nein, dann überspringe Löschteil
0562:  11 00 18   LD    DE,$1800   Zeiger auf Statuszeile
                                   (Zeile/Spalte)
0565:  CD 85 05   CALL  $0585      Setze Cursorpos
0568:  CD 48 0A   CALL  $0A48      Statuszeile löschen (40er)
056B:  CD 7A 07   CALL  $077A      Statuszeile löschen (VDC)
056E:  11 00 00   LD    DE,$0000   Auf erste Bildschirmposition
0571:  CD 85 05   CALL  $0585      Setze Cursorpos
0574:  CD 62 0A   CALL  $0A62      Cursorpos. bis Bildende löschen
0577:  DF         RST   $18        Dasselbe für VDC
0578:  20         .Byte $20        Sprungvektor 32

0579:  E6 80      AND   A,$80      Teste Bit 7
057B:  28 39      JR    Z,$05B6    Gelöscht: Zeichen normal ausg.
057D:  C1         POP   BC         Simulierte RS-Adresse zurück
057E:  E1         POP   HL         Zeiger zurück
057F:  5E         LD    E,(HL)     Hole Spalte
0580:  23         INC   HL         Zeiger auf nächstes Zeichen
0581:  E5         PUSH  HL         Rette Zeiger
0582:  C5         PUSH  BC         Rette simulierte CALL-Adresse
0583:  CB BA      RES   7,D        Bit 7 von Zeile löschen
0585:  D5         PUSH  DE         Rette Zeile/Spalte
0586:  CD BC 09   CALL  $09BC      40-Zeichen-Cursor setzen
0589:  D1         POP   DE         Hole Zeile/Spalte
058A:  DF         RST   $18        80-Zeichen-Cursor anspringen
058B:  04         .Byte $04        Sprungvektor ist 4

************************************ zu lesenden Block (Track/Sektor)
                                     anzeigen

058C:  11 4A 18   LD    DE,$184A   Zeile 24, Spalte 74 als Cursor
058F:  CD AB 06   CALL  $06AB      80-Zeichen-Cursorpos. setzen
0592:  11 22 18   LD    DE,$1822   Zeile 24, Spalte 34
0595:  CD BC 09   CALL  $09BC      40-Zeichen-Cursor setzen
0598:  3A 03 FD   LD    A,($FD03)  zu lesende Spur
059B:  CD A6 05   CALL  $05A6      in ASCII wandeln
059E:  16 20      LD    D,$20      Leerzeichen
05A0:  CD B6 05   CALL  $05B6      ausgeben
05A3:  3A 04 FD   LD    A,($FD04)  zu ladender Sektor

************************************ macht aus <Akku> ASCII

05A6:  06 2F      LD    B,$2F      ASCII "0" - 1
05A8:  04         INC   B          Zehnerstelle erhöhen
05A9:  D6 0A      SUB   $0A        Zieht (wenn möglich) 10 ab
05AB:  30 FB      JR    NC,$05A8   Zehnerstelle ist noch nicht null
05AD:  C6 3A      ADD   A,$3A      Fehlerkorrektur plus ASCII "0"
```

```
05AF:  F5          PUSH  AF         Rette Einerstelle
05B0:  78          LD    A,B        Zehnerstelle (ASCII) nach <Akku>
05B1:  CD B5 05    CALL  $05B5      und ausgeben
05B4:  F1          POP   AF         Hole Einerstelle (ASCII)

*********************************** Zeichen <Akku) ausgeben

05B5:  57          LD    D,A        Zweichen nach <D>
05B6:  D5          PUSH  DE         und merken
05B7:  CD 6E 09    CALL  $096E      Zeichen ausgeben
05BA:  D1          POP   DE         Hole auszugebendes Zeichen (VDC)
05BB:  DF          RST   $18        Auf 40-Zeichen-Bildschirm
05BC:  00          .Byte $00        Vektor 0

*********************************** Präpariere Zeichensatz 80 Zeichen

05BD:  21 04 30    LD    HL,$3004   Adresse im VDC-RAM
05C0:  CD 3D 09    CALL  $093D      Wert an $3004 holen
05C3:  04          INC   B          und auf 0 prüfen durch
05C4:  05          DEC   B          dekrementieren und inkr.
05C5:  C8          RET   Z          Null: wurde schon präpariert
05C6:  21 00 38    LD    HL,$3800   $3800 bis
05C9:  01 00 04    LD    BC,$0400   $3FFF mit dem
05CC:  16 00       LD    D,$00      Wert 0
05CE:  CD 47 08    CALL  $0847      füllen (löschen)
05D1:  21 A0 37    LD    HL,$37A0   ASCII 122
05D4:  11 A0 38    LD    DE,$38A0   wird
05D7:  01 08 00    LD    BC,$0008   ASCII 138
05DA:  CD B0 08    CALL  $08B0
05DD:  21 90 36    LD    HL,$3690   ASCII 105
05E0:  11 90 38    LD    DE,$3890   wird
05E3:  01 08 00    LD    BC,$0008   ASCII 137
05E6:  CD B0 08    CALL  $08B0
05E9:  21 E0 35    LD    HL,$35E0   ASCII 94 (PI)
05EC:  11 E0 38    LD    DE,$38E0   wird
05EF:  01 18 00    LD    BC,$0018   ASCII 95 (_)
05F2:  CD B0 08    CALL  $08B0
05F5:  21 10 30    LD    HL,$3010   A-Z (ASCII 1-26)
05F8:  11 10 36    LD    DE,$3610   nach
05FB:  01 98 01    LD    BC,$0198   ASCII 97 ff.
05FE:  CD B0 08    CALL  $08B0
0601:  21 00 30    LD    HL,$3000   $3000 bis $31FF
0604:  01 00 02    LD    BC,$0200   im VDC-RAM
0607:  16 00       LD    D,$00      löschen
0609:  CD 47 08    CALL  $0847
060C:  21 00 20    LD    HL,$2000   "@" (ASCII 64)
060F:  11 00 34    LD    DE,$3400   in VDC-RAM
0612:  01 08 00    LD    BC,$0008   kopieren
```

```
0615:  CD B0 08     CALL  $08B0
0618:  21 B0 21     LD    HL,$21B0      bis  (ASCII 27 bis 29)
061B:  11 B0 35     LD    DE,$35B0     nach
061E:  01 28 00     LD    BC,$0028     ASCII 91
0621:  CD B0 08     CALL  $08B0
0624:  21 C0 21     LD    HL,$21C0     ASCII 28 (Pfundzeichen)
0627:  11 00 38     LD    DE,$3800     nach
062A:  01 08 00     LD    BC,$0008     ASCII 128
062D:  CD B0 08     CALL  $08B0
0630:  21 E0 21     LD    HL,$21E0     ASCII 30 (^) wird 129
0633:  11 10 38     LD    DE,$3810     sowie
0636:  01 18 00     LD    BC,$0018     ASCII 31 (_) wird 130
0639:  CD B0 08     CALL  $08B0
063C:  21 00 24     LD    HL,$2400     Großbuchstaben und Sonderzeichen
063F:  11 00 3C     LD    DE,$3C00     nach $3C00 (ASCII 192)
0642:  01 F8 03     LD    BC,$03F8
0645:  CD B0 08     CALL  $08B0
0648:  11 1A 0F     LD    DE,$0F1A     ASCII 227
064B:  21 C0 35     LD    HL,$35C0     wird
064E:  CD 70 06     CALL  $0670        ASCII 92
0651:  21 E0 35     LD    HL,$35E0     ASCII 228-230
0654:  06 03        LD    B,$03        wird
0656:  CD 62 06     CALL  $0662        ASCII 94-96
0659:  21 B0 37     LD    HL,$37B0     ASCII 231-235
065C:  06 05        LD    B,$05        wird
065E:  18 02        JR    $0662        ASCII 123 ff.
0660:  E1           POP   HL           Hole Rücksprungadresse nach HL
0661:  E3           EX    (SP),HL      Eine Rücksprungadresse löschen

************************************ Kopiere (DE)->(HL) VDC (BC) mal

0662:  C5           PUSH  BC           Zähler retten
0663:  E5           PUSH  HL           Zieladresse merken
0664:  CD 70 06     CALL  $0670        (DE)->(HL) VDC-RAM, 8 Bytes
0667:  E1           POP   HL           Zieladresse zurückholen
0668:  01 10 00     LD    BC,$0010     Und 16 addieren (anstatt 8)
066B:  09           ADD   HL,BC        wegen internen VDC-Aufbau
066C:  C1           POP   BC           Hole Zähler zurück
066D:  10 F3        DJNZ  $0662        noch ein Zeichen zu kopieren?
066F:  C9           RET                Nein, dann Ende

************************************ (DE)->(HL) VDC; 8 Bytes

0670:  CD 53 09     CALL  $0953        HL als Update anmelden
0673:  26 08        LD    H,$08        8 Bytes sollen kopiert werden
0675:  1A           LD    A,(DE)       Hole das Zeichen aus RAM
0676:  ED 79        OUT   (C),A        und speichere es nach VDC
0678:  0D           DEC   C            Zeiger auf Status-Flag
```

```
0679:  13          INC   DE           Zeiger auf Tabelle erhöhen
067A:  ED 78       IN    A,(C)        Hole Status-Flag
067C:  17          RLA                Teste Ready-Bit
067D:  30 FB       JR    NC,$067A     Noch nicht fertig
067F:  0C          INC   C            Zeiger wieder auf $D601
0680:  25          DEC   H            Erniedrige Zähler
0681:  20 F2       JR    NZ,$0675     Springe, wenn noch keine 8 Byte
0683:  C9          RET                sonst beende Routine

************************************ <A> Zeichen ausgeben inkl.
                                      Cursorbewegung

0684:  2A 11 24    LD    HL,($2411)   Cursoradresse holen
0687:  CD 07 09    CALL  $0907        Zeichen ausgeben auf 80-Zeichen
068A:  3A 13 24    LD    A,($2413)    Cursorspalte
068D:  FE 4F       CP    $4F          rechten Rand (79) erreicht?
068F:  28 3C       JR    Z,$06CD      Ja, dann nächste Zeile
0691:  3C          INC   A            sonst erhöhe den Spaltenzeiger
0692:  32 13 24    LD    ($2413),A    und merke die neue Spalte
0695:  2A 11 24    LD    HL,($2411)   Hole Cursoradresse
0698:  23          INC   HL           und ebenfalls um eine Stelle
0699:  22 11 24    LD    ($2411),HL   erhöhen und wieder abspeichern

************************************ HL als Cursoradresse setzen

069C:  3E 0E       LD    A,$0E        Cursoradresse Hi-Byte
069E:  CD 45 09    CALL  $0945        anmelden bei VDC
06A1:  ED 61       OUT   (C),H        Higher Byte an VDC übergeben
06A3:  3E 0F       LD    A,$0F        Register 15 ist Cursoradresse Lo
06A5:  CD 45 09    CALL  $0945        anmelden bei VDC
06A8:  ED 69       OUT   (C),L        und auch Lo-Byte übergeben
06AA:  C9          RET                Ende der Übertragung

************************************ 80-Zeichen-Cursorposition setzen
                                      D:Spalte, E:Zeile

06AB:  7A          LD    A,D          Hole Zeile
06AC:  FE 19       CP    $19          Größer als 24
06AE:  D0          RET   NC           Ja, dann ungültig und Ende
06AF:  7B          LD    A,E          Hole Spalte
06B0:  FE 50       CP    $50          größer als 79?
06B2:  D0          RET   NC           Ja, dann ungültig und Ende
06B3:  EB          EX    DE,HL        zwecks Speicherung HL nach DE
06B4:  22 13 24    LD    ($2413),HL   und Cursorposition merken

06B7:  2A 13 24    LD    HL,($2413)   Cursorposition holen
06BA:  CD CE 0C    CALL  $0CCE        Zeile*80 + Spalte
06BD:  22 11 24    LD    ($2411),HL   Adresse merken
```

```
06C0:  18 DA       JR    $069C        Cursoradresse an VDC übergeben

*********************************** Zeile um eins erniedrigen

06C2:  3A 14 24    LD    A,($2414)    Hole Cursorzeile
06C5:  B7          OR    A            setze die CPU-Flags
06C6:  C8          RET   Z            Zeile bereits erste, dann Schluß
06C7:  3D          DEC   A            sonst erniedrige die Zeile
06C8:  32 14 24    LD    ($2414),A    und merke diese
06CB:  18 EA       JR    $06B7        neue Cursoradresse berechnen

*********************************** Spalte=0 (erste Spalte) setzen
                                      inkl. Zeilenbeeinflussung

06CD:  AF          XOR   A            Akku wird null
06CE:  32 13 24    LD    ($2413),A    und als Spalte merken
06D1:  3A 14 24    LD    A,($2414)    Hole aktuelle Cursorzeile
06D4:  FE 17       CP    $17          Ist Zeile die 23.?
06D6:  28 21       JR    Z,$06F9      Ist erreicht, dann Sprung
06D8:  30 1A       JR    NC,$06F4     Zeile ist 24
06DA:  3C          INC   A            sonst Zeile um eins erhöhen
06DB:  18 EB       JR    $06C8        und Zeile merken

*********************************** Spalte um eins erniedrigen

06DD:  3A 13 24    LD    A,($2413)    Hole aktuelle Spalte
06E0:  B7          OR    A            setze Flags zum Testen auf null
06E1:  C8          RET   Z            Bereits erste Spalte, dann Ende
06E2:  3D          DEC   A            Erniedrige den Spaltenzeiger
06E3:  32 13 24    LD    ($2413),A    und merke neue Spalte
06E6:  18 CF       JR    $06B7        Neue Cursoradresse berechnen

*********************************** Spalte um eins erhöhen

06E8:  3A 13 24    LD    A,($2413)    Hole aktuelle Spalte
06EB:  3C          INC   A            und um eins erhöhen
06EC:  FE 50       CP    $50          ist Spalte 80 erreicht?
06EE:  20 F3       JR    NZ,$06E3     Nein, dann merke Spalte
06F0:  C9          RET                sonst beende Routine

*********************************** Spalte auf 0 (erste Spalte) setzen

06F1:  AF          XOR   A            Akku gleich Null setzen
06F2:  18 EF       JR    $06E3        und dann als neue Spalte merken

*********************************** Zeile=23 setzen

06F4:  3E 17       LD    A,$17        Zeile auf 23. setzen
```

```
06F6:  32 14 24    LD    ($2414),A   und im Speicher merken
06F9:  21 50 00    LD    HL,$0050    Bildschirm um eine
06FC:  11 00 00    LD    DE,$0000    Zeile nach oben scrollen
06FF:  01 30 07    LD    BC,$0730    durch Kopieren der 2 in die
0702:  CD B0 08    CALL  $08B0       erste Zeile etc.
0705:  21 30 07    LD    HL,$0730    Zeiger auf letzte Zeile
0708:  01 50 00    LD    BC,$0050    Anzahl ist 80 Zeichen
070B:  CD 41 08    CALL  $0841       Zeile löschen
070E:  18 A7       JR    $06B7       und neue Cursorposition anmelden

************************************ Neues Attribut definieren (B:zu
                                     löschende, C:zu setzende Attr.)

0710:  3A 15 24    LD    A,($2415)   Hole Attribut
0713:  2F          CPL               komplementiere Attribut
0714:  B0          OR    B           zu löschende Bits (Attr.)
0715:  2F          CPL               zurückkomplementieren
0716:  B1          OR    C           zu setzende Bits (Eigenschaften)
0717:  32 15 24    LD    ($2415),A   Abspeichern des neuen Attributes
071A:  C9          RET               Ende der Routine

************************************

071B:  78          LD    A,B         Zeichen holen
071C:  D6 20       SUB   $20         ASCII 32 subtrahieren (Leer)
071E:  FE 20       CP    $20         ASCII 32?
0720:  38 0A       JR    C,$072C     Kleiner, dann Sprung
0722:  0E 20       LD    C,$20       Merker für ASCII 32 abgezogen
0724:  CD E5 0C    CALL  $0CE5       ASCII + Code umwandeln
0727:  D8          RET   C           Geschafft
0728:  7E          LD    A,(HL)      Hole Zeichen aus Tabelle
0729:  E6 0F       AND   $0F         Bits 4-7 ausmaskieren
072B:  80          ADD   A,B         und <B> als Offset dazu
072C:  32 00 24    LD    ($2400),A   merke Zeichen
072F:  0E 20       LD    C,$20       ASCII 32 subtrahiert-Zeiger
0731:  C6 30       ADD   A,$30       ASCII 48 ("0") addieren
0733:  21 0A 0F    LD    HL,$0F0A    Tabelle für Farvanpassung
0736:  CD E8 0C    CALL  $0CE8       umwandeln
0739:  7E          LD    A,(HL)      Hole Farbwert
073A:  80          ADD   A,B         und addiere Attribut
073B:  FE 10       CP    $10         Übertrag ins höherw. Nibble?
073D:  38 1B       JR    C,$075A     Vordergrundfarbe wird definiert
073F:  E6 0F       AND   $0F         sonst Bits 4-7 ausmaskieren
0741:  32 16 24    LD    ($2416),A   Hintergrundfarbe merken
0744:  F5          PUSH  AF          Rette Farbcode
0745:  3E 1A       LD    A,$1A       Register
                                     26=Vorder/Hintergrundfarbe
0747:  CD 45 09    CALL  $0945       VDC-Status abwarten und anmelden
```

```
074A:  F1          POP   AF          Hole Farbewert
074B:  ED 79       OUT   (C),A       und Farbe setzen
074D:  C9          RET               Ende der Routine

************************************ B:Backgnd, D:Attribut A:Foregnd

074E:  3A 16 24    LD    A,($2416)   Hintergrundfarbe holen
0751:  47          LD    B,A         und nach <B>
0752:  3A 15 24    LD    A,($2415)   Attribut holen
0755:  57          LD    D,A         und nach D
0756:  3A 17 24    LD    A,($2417)   Vordergrundfarbe holen
0759:  C9          RET               Ende der Routine

************************************ Neue Vordergrundfarbe definieren

075A:  47          LD    B,A         Farbe nach B
075B:  3A 15 24    LD    A,($2415)   Hole aktuelles Attribut
075E:  E6 F0       AND   $F0         Farbennibble ausmaskieren
0760:  B0          OR    B           und neue Farbe setze
0761:  32 15 24    LD    ($2415),A   neues Attribut merken
0764:  3A 00 24    LD    A,($2400)   Hole Hintergrundfarbe und
0767:  32 17 24    LD    ($2417),A   merken
076A:  2A 11 24    LD    HL,($2411)  Hole Cursoradresse
076D:  11 00 08    LD    DE,$0800    und Offset für Attribut-RAM
0770:  19          ADD   HL,DE       addieren
0771:  CD 53 09    CALL  $0953       HL als Update anmelden
0774:  3A 15 24    LD    A,($2415)   Attribut holen
0777:  ED 79       OUT   (C),A       und an Cursoradresse speichern
0779:  C9          RET               Ende der Routine

************************************ Cursorpos. bis Zeilenende löschen

077A:  CD C7 0C    CALL  $0CC7       Hole Cursorpos., Anzahl Zeichen
077D:  03          INC   BC          erhöhe die Anzahl
077E:  18 0E       JR    $078E       und lösche den Rest der Zeile

************************************ Cursorpos. bis Bildende löschen

0780:  CD C7 0C    CALL  $0CC7       Hole Cursorpos., Anzahl Zeichen
0783:  EB          EX    DE,HL       Cursoradresse nach DE
0784:  21 80 07    LD    HL,$0780    Adresse Anfang der Statuszeile
0787:  AF          XOR   A           Lösche Carry für Subtraktion
0788:  ED 52       SBC   HL,DE       Anzahl Zeichen bis Statuszeile
078A:  F8          RET   M           Wenn negativ, dann Ende (Fehler)
078B:  44          LD    B,H         Sonst BC gleich
078C:  4D          LD    C,L         Anzahl Zeichen bis Statuszeile
078D:  EB          EX    DE,HL       und aktuelle Cursorposition
                                     wieder nach HL
```

```
078E:  C3 41 08     JP    $0841        Lösche bis Anfang Statuszeile

*********************************** 1 Zeichen einfügen
                                    Restzeile verschieben

0791:  CD C7 0C     CALL  $0CC7        Hole Cursorpos. & Anzahl Zeichen
0794:  21 4F 00     LD    HL,$004F     addiere 79 zu Zeilenanfang
0797:  19           ADD   HL,DE        zum Zeilenanfang
0798:  3D           DEC   A            Anzahl Zeichen bis Zeilenende-1
0799:  28 2C        JR    Z,$07C7      keines mehr, dann Schluß
079B:  54           LD    D,H          Adresse des Zeilenende
079C:  5D           LD    E,L          nach DE
079D:  2B           DEC   HL           Adresse Zeilenende-1
079E:  C5           PUSH  BC           Rette Anzahl
079F:  E5           PUSH  HL           Rette Quelle
07A0:  D5           PUSH  DE           Rette Ziel
07A1:  CD AD 07     CALL  $07AD        Kopiere (HL)->(DE) in VDC-RAM
07A4:  01 00 08     LD    BC,$0800     Addiere nun Offset für
07A7:  E1           POP   HL           Attribut-RAM im VDC
07A8:  09           ADD   HL,BC        zur Quelladresse
07A9:  EB           EX    DE,HL        und in DE merken
07AA:  E1           POP   HL           Hole Zieladresse von Stack
07AB:  09           ADD   HL,BC        ebenfalls den Offset addieren
07AC:  C1           POP   BC           Anzahl holen

07AD:  C5           PUSH  BC           und wieder sichern
07AE:  CD 53 09     CALL  $0953        HL als Update anmelden
07B1:  ED 78        IN    A,(C)        Hole aktuellen Inhalt
07B3:  EB           EX    DE,HL        Ziel und Quelle vertauschen
07B4:  F5           PUSH  AF           Rette das ermittelte Zeichen
07B5:  CD 53 09     CALL  $0953        HL wieder als Update anmelden
07B8:  F1           POP   AF           ermitteltes Zeichen zurückholen
07B9:  ED 79        OUT   (C),A        und in Zieladresse kopieren
07BB:  EB           EX    DE,HL        Ziel und Quelle wieder ok.
07BC:  C1           POP   BC           Hole Anzahl zurück
07BD:  2B           DEC   HL           Erniedrige den Quellzeiger
07BE:  1B           DEC   DE           Erniedrige den Zielzeiger
07BF:  0B           DEC   BC           Erniedrige den Zähler
07C0:  78           LD    A,B          Feststellen, ob Register-
07C1:  B1           OR    C            paar BC gleich null ist
07C2:  20 E9        JR    NZ,$07AD     nein, nächstes Zeichen kopieren
07C4:  2A 11 24     LD    HL,($2411)   sonst Cursoradresse holen
07C7:  C3 05 09     JP    $0905        und ein Leerzeichen ausgeben

*********************************** Zeichen an Cursorposition löschen

07CA:  CD C7 0C     CALL  $0CC7        Hole Cursoradr. & Anzahl Zeichen
07CD:  D5           PUSH  DE           Zeilenanfang auf Stack sichern
```

```
07CE:  54          LD    D,H          aktuelle Cursorposition
07CF:  5D          LD    E,L          nach DE kopieren
07D0:  23          INC   HL           Quelle um eins erhöhen
07D1:  CD B0 08    CALL  $08B0        (HL)->(DE) BC mal = 1 Zeichen
                                      löschen
07D4:  E1          POP   HL           Zeilenanfang nach HL
07D5:  11 4F 00    LD    DE,$004F     und 79 für Zeilenende hinzu-
07D8:  19          ADD   HL,DE        addieren
07D9:  C3 05 09    JP    $0905        Zeilenende ein <Space> ausgeben

************************************ An Cursorzeile eine Zeile einfügen

07DC:  11 62 0F    LD    DE,$0F62     Zeiger auf Tabelle
07DF:  3E 17       LD    A,$17        Zeile 23 in Akku
07E1:  2A 13 24    LD    HL,($2413)   Hole Zeile/Spalte
07E4:  BC          CP    H            Zeile 23 erreicht?
07E5:  CA 05 07    JP    Z,$0705      Ja, dann lösche 23. Zeile
07E8:  38 1A       JR    C,$0804      kleiner als 23, dann Sprung
07EA:  21 E0 06    LD    HL,$06E0     (vorvorletzte Zeile)
07ED:  11 30 07    LD    DE,$0730     Zieladresse (vorletzte Zeile)
07F0:  06 18       LD    B,$18        24 Zeilen sind zu kopieren
07F2:  CD 0A 08    CALL  $080A        Zeile (HL) nach (DE) kopieren
07F5:  3A 14 24    LD    A,($2414)    Hole Zeile
07F8:  B8          CP    B            aktuelle Zeile erreicht?
07F9:  20 F7       JR    NZ,$07F2     Nein, dann weiterkopieren
07FB:  CD C7 0C    CALL  $0CC7        Hole Cursorpos. & Anzahl Zeichen
07FE:  EB          EX    DE,HL        HL:Zeilenanfang
07FF:  01 50 00    LD    BC,$0050     80 als Anfang Folgezeile
0802:  18 3D       JR    $0841        Cursorzeile löschen

0804:  3C          INC   A            Diese Stelle wird nie
0805:  BD          CP    L            erreicht, da sich der Cursor
0806:  C0          RET   NZ           dann in der Statuszeile befände
0807:  C3 2C 05    JP    $052C        Text ab DE ausgeben

************************************ Zeile (HL) -> (DE) kopieren im VDC

080A:  C5          PUSH  BC           Anzahl auf Stack sichern
080B:  E5          PUSH  HL           Quelladresse auf Stack sichern
080C:  D5          PUSH  DE           Zieladresse auf Stack sichern
080D:  01 50 00    LD    BC,$0050     80 Zeichen pro Zeile
0810:  CD B0 08    CALL  $08B0        Zeile kopieren
0813:  01 B0 FF    LD    BC,$FFB0     Komplement von 80 addieren
                                      ergibt Subtraktion
0816:  E1          POP   HL           Hole Zieladresse
0817:  09          ADD   HL,BC        subtrahiere 80 Zeichen
0818:  EB          EX    DE,HL        und nach DE
0819:  E1          POP   HL           Hole Quelladresse
```

```
081A:  09          ADD   HL,BC        ebenfalls 80 subtrahieren
081B:  C1          POP   BC           Hole Anzhal
081C:  05          DEC   B            den Zähler lediglich erniedrigen
081D:  C9          RET                und Ende der Routine

************************************ Cursorz. löschen, Rest raufziehen

081E:  3A 14 24    LD    A,($2414)    Hole Zeile
0821:  FE 18       CP    $18          Ist Zeile 24 erreicht?
0823:  D0          RET   NC           Ja, dann Schluß
0824:  CD C7 0C    CALL  $0CC7        Cursorpos. und Anzhal Zeichen
0827:  21 50 00    LD    HL,$0050     80 zur Startadresse der Cursor-
082A:  19          ADD   HL,DE        zeile hinzuaddieren
082B:  EB          EX    DE,HL        und in DE merken
082C:  E5          PUSH  HL           Rette Startadresse auf Stack
082D:  21 80 07    LD    HL,$0780     Startadresse der Statuszeile
0830:  AF          XOR   A            Lösche Carry für Subtraktion
0831:  ED 52       SBC   HL,DE        Min. Startadresse der Folgezeile
0833:  44          LD    B,H          ergibt Anzahl Zeichen bis
0834:  4D          LD    C,L          Bildschirmende nach BC
0835:  E1          POP   HL           Hole Startadresse zurück
0836:  EB          EX    DE,HL        Vertausche Quelle und Ziel
0837:  CD B0 08    CALL  $08B0        (HL)->(DE) im VDC-RAM
083A:  C3 05 07    JP    $0705        Zeile vor Statuszeile löschen
```

```
************************************** VDC-RAM mit Wert füllen

083D:  E1          POP   HL          Rücksprungadresse holen
083E:  E3          EX    (SP),HL     und vorhergehende
                                     Rücksprungadresse löschen
083F:  18 06       JR    $0847       Sprung nach Routine

0841:  3A 15 24    LD    A,($2415)   Hole Attribut
0844:  5F          LD    E,A         Attribut in E merken
0845:  16 20       LD    D,$20       ASCII-Code für Leerzeichen

************************************** (HL) in VDC-RAM mit D füllen,
                                       Attribut mit E
                                       (HL+800) wird nur ggf. gefüllt

0847:  78          LD    A,B         Hi-Byte von Anzahl
0848:  A7          AND   A           und Hi-Byte auf null testen
0849:  28 0D       JR    Z,$0858     Null, dann nur Lo-Byte füllen
084B:  E5          PUSH  HL          Rette Zieladresse
084C:  D5          PUSH  DE          Rette Füllwerte
084D:  C5          PUSH  BC          Rette Anzahl
084E:  AF          XOR   A           Akku=0 bedeutet 256 Zeichen
084F:  CD 5B 08    CALL  $085B       Fülle (HL) mit D, 256 mal
0852:  C1          POP   BC          Hole Anzahl zurück
0853:  D1          POP   DE          Hole Füllwerte zurück
0854:  E1          POP   HL          Hole Zieladresse zurück
0855:  24          INC   H           Hi-Byte erhöhen
0856:  10 F3       DJNZ  $084B       falls > 256 Zeichen, dann Sprung
0858:  79          LD    A,C         Teste Lo-Byte auf
0859:  A7          AND   A           null (keine mehr zu füllen)
085A:  C8          RET   Z           und beende, wenn Schluß

************************************** <D> an <HL> <a> mal speichern

085B:  F5          PUSH  AF          Rette Anzahl
085C:  E5          PUSH  HL          Rette Zieladresse
085D:  D5          PUSH  DE          Rette Füllzeichen
085E:  CD 6C 08    CALL  $086C       Zeichen D speichern
0861:  D1          POP   DE          Hole Füllzeichen
0862:  01 00 08    LD    BC,$0800    Offset für Attribut-RAM
0865:  E1          POP   HL          Zieladresse zurückholen
0866:  09          ADD   HL,BC       und Offset addieren
0867:  CD FE 08    CALL  $08FE       HL auf gültige Adresse abtesten
086A:  F1          POP   AF          wenn ok, dann hole Zähler
086B:  53          LD    D,E         und Attribut als Füllzeichen
```

```
************************************ <D> <A> mal an <HL> im VDC-RAM

086C:  F5        PUSH  AF         Rette Zähler auf Stack
086D:  CD 53 09  CALL  $0953      HL als Update anmelden
0870:  ED 51     OUT   (C),D      und Füllzeichen übergeben
0872:  F1        POP   AF         hole Anzahl von Stack
0873:  3D        DEC   A          erniedrige den Zähler
0874:  C8        RET   Z          und beende, wenn genug gefüllt
0875:  F5        PUSH  AF         sonst rette Zähler
0876:  3E 18     LD    A,$18      Register 24 (Copy-Bit)
0878:  CD 45 09  CALL  $0945      anmelden
087B:  ED 78     IN    A,(C)      Registerinhalt holen
087D:  E6 7F     AND   $7F        und Copy-Bit ausmaksieren
087F:  ED 79     OUT   (C),A      wieder in VDC-Speicher
0881:  3E 1E     LD    A,$1E      Reg. 31 (Wordcount) auswählen
0883:  CD 45 09  CALL  $0945      und anmelden
0886:  F1        POP   AF         hole Anzahl von Stack
0887:  ED 79     OUT   (C),A      und Restanzahl an VDC übergeben
0889:  06 00     LD    B,$00      Hi-Byte von BC Null setzen
088B:  4F        LD    C,A        und Lo-Byte mit Restanzahl
088C:  03        INC   BC         addiere eins
088D:  09        ADD   HL,BC      addiere Startadresse
088E:  D5        PUSH  DE         rette Startadresse
088F:  E5        PUSH  HL         rette errechnete Schlußadresse
0890:  3E 12     LD    A,$12      Update Hi-Byte-Register
0892:  CD 45 09  CALL  $0945      anmelden
0895:  ED 60     IN    H,(C)      und Wert auslesen
0897:  3E 13     LD    A,$13      Update Lo-Byte-Register
0899:  CD 45 09  CALL  $0945      anmelden
089C:  ED 68     IN    L,(C)      und Wert auslesen
089E:  D1        POP   DE         hole errechnete Schlußadresse
089F:  C1        POP   BC         hole Startadresse
08A0:  CD FA 00  CALL  $00FA      Vergleiche HL mit DE
08A3:  D0        RET   NC         Alles klar, kein Fehler!
08A4:  C5        PUSH  BC         Anzahl auf Stack
08A5:  CD 53 09  CALL  $0953      HL als Update anmelden
08A8:  C1        POP   BC         hole Restanzahl
08A9:  ED 41     OUT   (C),B      und Fehlerkorrektur
08AB:  23        INC   HL         falls errechnete Schlußadresse
08AC:  18 F2     JR    $08A0      und tatsächliche Schlußadresse
08AE:  E1        POP   HL         ungleich Rücksprungadresse holen
08AF:  E3        EX    (SP),HL    und um eins tiefer auf Stack

************************************ (HL) -> (DE) im VDC-RAM <BC> mal

08B0:  78        LD    A,B        Hole Hi-Byte von Anzahl
08B1:  A7        AND   A          uns teste Hi-Byte auf null
08B2:  28 0E     JR    Z,$08C2    Wenn null, dann Anzahl<256
```

```
08B4:  E5          PUSH  HL          rette Quelladresse
08B5:  D5          PUSH  DE          Rette Zieladresse
08B6:  C5          PUSH  BC          Rette Anzahl auf Stack
08B7:  AF          XOR   A           Lösche Akku für 256 Zeichen
08B8:  CD C5 08    CALL  $08C5       (HL)->(DE) <A> mal
08BB:  C1          POP   BC          Hole Anzahl
08BC:  D1          POP   DE          Hole Zieladresse
08BD:  E1          POP   HL          Hole Quelladresse
08BE:  24          INC   H           Hi-Byte der Quelladresse erhöhen
08BF:  14          INC   D           Hi-Byte der Zeiladresse erhöhen
08C0:  10 F2       DJNZ  $08B4       > 256 Zeichen, dann weiter
08C2:  79          LD    A,C         Hole Lo-Anzahl
08C3:  A7          AND   A           teste auf null
08C4:  C8          RET   Z           und Schluß, wenn null
08C5:  EB          EX    DE,HL       sonst vertausche Quell- und Ziel
08C6:  F5          PUSH  AF          Rette Anzahl auf Stack
08C7:  E5          PUSH  HL          Rette Zieladresse
08C8:  D5          PUSH  DE          Rette Quelladresse
08C9:  CD D8 08    CALL  $08D8       (DE)->(HL) im VDC-RAM <A> mal
08CC:  01 00 08    LD    BC,$0800    Offset für Attribut-RAM
08CF:  E1          POP   HL          Hole Quelladresse
08D0:  09          ADD   HL,BC       addiere Offset
08D1:  EB          EX    DE,HL       Quelladresse+Offset nach DE
08D2:  E1          POP   HL          Hole Zieladresse
08D3:  09          ADD   HL,BC       addiere Offset
08D4:  CD FE 08    CALL  $08FE       Teste auf Speichergrenzen
08D7:  F1          POP   AF          Hole Anzahl

************************************ (DE) -> (HL) im VDC <A> mal

08D8:  F5          PUSH  AF          Rette Anzahl auf Stack
08D9:  CD 53 09    CALL  $0953       Melde HL als Updateadresse an
08DC:  3E 18       LD    A,$18       Register 24 (Copy-Bit)
08DE:  CD 45 09    CALL  $0945       anmelden
08E1:  ED 78       IN    A,(C)       Hole Registerinhalt
08E3:  F6 80       OR    $80         und setze das Copybit
08E5:  ED 79       OUT   (C),A       Register an VDC mitteilen
08E7:  3E 20       LD    A,$20       Register 32 (Block-Start-Hi)
08E9:  CD 45 09    CALL  $0945       im VDC anmelden
08EC:  ED 51       OUT   (C),D       Hi-Adresse Quelle übergeben
08EE:  3E 21       LD    A,$21       Register 33 (Block-Start-Lo)
08F0:  CD 45 09    CALL  $0945       in VDC anmelden
08F3:  ED 59       OUT   (C),E       und Lo-Adresse Quelle übergeben
08F5:  3E 1E       LD    A,$1E       Register 31 (Wordcount)
08F7:  CD 45 09    CALL  $0945       anmelden
08FA:  F1          POP   AF          und Anzahl von Stack holen
08FB:  ED 79       OUT   (C),A       Anzahl VDC mitteilen
```

```
08FD:  C9           RET                  Ende der Routine

************************************ Testet, ob nach Addierung des
                                     Offset <HL> auf
                                     Attribut-RAM zeigt.

08FE:  7C           LD     A,H           Hi-Byte nach Akku holen
08FF:  FE 20        CP     $20           und auf Grenze checken
0901:  D8           RET    C             Wenn Carry gesetzt, ist <HL> ok
0902:  F1           POP    AF            Hole AF von Stack
0903:  F1           POP    AF            Hole Rücksprungadresse von Stack
0904:  C9           RET                  Rücksprung nach CALL $085B

************************************ Leerzeichen an (HL) ausgeben (VDC)

0905:  16 20        LD     D,$20         ASCII-Wert für <Space>
0907:  3A 15 24     LD     A,($2415)     Attribut holen

************************************ <D> mit Attr. <A> an (HL) ausgeben

090A:  E5           PUSH   HL            Rette Zieladresse
090B:  D5           PUSH   DE            Rette Zeichen/Attribut
090C:  11 00 08     LD     DE,$0800      Offset für Attribut-RAM
090F:  19           ADD    HL,DE         auf Zieladresse addieren
0910:  57           LD     D,A           Attribut als Füllzeichen
0911:  CD 16 09     CALL   $0916         <D> an (HL) ausgeben
0914:  D1           POP    DE            Hole Zeichen
0915:  E1           POP    HL            Hole Zieladresse
0916:  CD 53 09     CALL   $0953         HL als Update anmelden
0919:  ED 51        OUT    (C),D         und Zeichen an (HL) ausgeben
091B:  C9           RET                  Ende der Routine

************************************ Hole <C>:Attribut, <B>:Zeichen, an
                                     Cursorpos. <DE>

091C:  CD AB 06     CALL   $06AB         Cursorposition <DE> setzen
091F:  2A 11 24     LD     HL,($2411)    Hole Cursoradresse
0922:  CD 33 09     CALL   $0933         Hole Zeichen/Attribut
0925:  4F           LD     C,A           <C> ist Attribut
0926:  C9           RET                  Ende der Routine

************************************ <B>:Zeichen, <C>:Attribut, an <DE>

0927:  C5           PUSH   BC            Rette Zeichen/Attribut
0928:  CD AB 06     CALL   $06AB         Cursorposition <DE> setzen
092B:  2A 11 24     LD     HL,($2411)    Hole Cursoradresse
092E:  C1           POP    BC            Hole Zeichen/Atrribut
092F:  50           LD     D,B           <D> ist Zeichen
```

```
0930:  79          LD    A,C         <A> ist Attribut
0931:  18 D7       JR    $090A       Zeichen und Attribut ausgeben

*********************************** <A>:Attribut, <B>:Zeichen von (HL)

0933:  E5          PUSH  HL          Rette Adresse
0934:  11 00 08    LD    DE,$0800    Offset für Attributadresse
0937:  19          ADD   HL,DE       addieren
0938:  CD 3D 09    CALL  $093D       Hole Wert an VDC-Adresse (HL)
093B:  78          LD    A,B         Attribut nach <A>
093C:  E1          POP   HL          Hole Textadresse
093D:  F5          PUSH  AF          Rette Attribut
093E:  CD 53 09    CALL  $0953       (HL) als Update anmelden
0941:  F1          POP   AF          Hole Attribut
0942:  ED 40       IN    B,(C)       Hole Wert an Adresse (HL)
0944:  C9          RET               Ende der Routine

*********************************** VDC-Status abwarten und anw.

0945:  F5          PUSH  AF          Rette auszugebendes Register
0946:  01 00 D6    LD    BC,$D600    Startadresse VDC-Chip
0949:  ED 78       IN    A,(C)       Hole Status
094B:  17          RLA               Shifte Status-Bit ins Carry
094C:  30 FB       JR    NC,$0949    Noch nicht fertig -> Sprung
094E:  F1          POP   AF          Hole Register wieder von Stack
094F:  ED 79       OUT   (C),A       und Register anwählen
0951:  0C          INC   C           Zeige auf $D601
0952:  C9          RET               und RETurn aus Routine

*********************************** HL als Update-Adresse

0953:  3E 12       LD    A,$12       Update-Adresse Hi
0955:  CD 45 09    CALL  $0945       anmelden
0958:  ED 61       OUT   (C),H       Hi-Byte übergeben
095A:  3E 13       LD    A,$13       Update-Adresse Lo
095C:  CD 45 09    CALL  $0945       anmelden
095F:  ED 69       OUT   (C),L       Lo-Byte übergeben
0961:  3E 1F       LD    A,$1F       Wordcount-Register
0963:  CD 45 09    CALL  $0945       anmelden
0966:  0D          DEC   C           Zeiger wieder auf $D600
0967:  ED 78       IN    A,(C)       Hole Status
0969:  17          RLA               Rolle Status ins Carry
096A:  30 FB       JR    NC,$0967    Noch nicht fertig
096C:  0C          INC   C           Jetzt ja, Zeiger auf $D601
096D:  C9          RET               RETurn aus Unterroutine
```

```
************************************ Zeichen <D> auf 40-Zeichen-Schirm

096E:  42         LD    B,D          Zeichen nach <B>
096F:  CD 7F 0C   CALL  $0C7F        ASCII-Codeumwandlung VIC
0972:  2A 09 24   LD    HL,($2409)   80-Zeichen-Adresse
0975:  47         LD    B,A          Zeichen nach <B>
0976:  3A 10 24   LD    A,($2410)    Zeichenoffset holen (Bit 7 1/0)
0979:  B0         OR    B            mit Zeichen verknüpfen
097A:  77         LD    (HL),A       und ins RAM schreiben
097B:  23         INC   HL           Zeiger erhöhen
097C:  22 09 24   LD    ($2409),HL   und merken
097F:  11 FF 07   LD    DE,$07FF     Offset für Farbram
0982:  19         ADD   HL,DE        hinzuaddieren
0983:  3A 0D 24   LD    A,($240D)    Hole Zeichenfarbe
0986:  77         LD    (HL),A       und Zeichenfarbe setzen
0987:  3A 0B 24   LD    A,($240B)    Hole Cursorposition
098A:  FE 4F      CP    $4F          Letzte Spalte?
098C:  28 5F      JR    Z,$09ED      Ja, dann Zeilensprung
098E:  3C         INC   A            sonst erhöhe den Spaltenzeiger
098F:  32 0B 24   LD    ($240B),A    und merke neue Position
0992:  C3 4A 0C   JP    $0C4A        Zeile darstellen

************************************ Hole Zeichen <B> und Farbe <C> von
                                     Cursorpos (DE)

0995:  CD C1 09   CALL  $09C1        Zeile/Spalte definieren (DE)
0998:  2A 09 24   LD    HL,($2409)   Cursoradr. 80-Zeichen-Simulator
099B:  46         LD    B,(HL)       Hole Zeichen an Cursorposition
099C:  11 00 08   LD    DE,$0800     Offset für Attribut-RAM
099F:  19         ADD   HL,DE        addieren
09A0:  4E         LD    C,(HL)       Hole Attribut an Cursorposition
09A1:  C9         RET                Ende der Routine

************************************ <B>:Zeichen, <C>:Attribut an (DE)

09A2:  C5         PUSH  BC           Rette Zeichen/Attribut
09A3:  CD C1 09   CALL  $09C1        Setze Cursorposition
09A6:  C1         POP   BC           Hole Zeichen/Attribut
09A7:  2A 09 24   LD    HL,($2409)   80-Zeichen-Simulator-Adresse
09AA:  78         LD    A,B          Zeichen in <Akku>
09AB:  E6 7F      AND   $7F          Bit 7 löschen
09AD:  CB 71      BIT   6,C          Bit 6 testen
09AF:  28 02      JR    Z,$09B3      Ist ungesetzt
09B1:  C6 80      ADD   A,$80        Bit 7 setzen (reverses Zeichen)
09B3:  77         LD    (HL),A       und Zeichen setzen
09B4:  11 00 08   LD    DE,$0800     Offset für Attribut-RAM
09B7:  19         ADD   HL,DE        addieren
09B8:  71         LD    (HL),C       Attribut ebenfalls definieren
```

```
09B9:  C3 4A 0C     JP    $0C4A         Zeile darstellen

************************************ Def. Zeile/Spalte

09BC:  21 04 24     LD    HL,$2404      Adresse 40-Zeichen-Anpassung
09BF:  CB F6        SET   6,(HL)        40-Zeichen-Bit setzen
09C1:  7A           LD    A,D           Hole Zeile
09C2:  FE 19        CP    $19           Größer als 24?
09C4:  D0           RET   NC            Ja, dann Ende (Fehler)
09C5:  7B           LD    A,E           Hole Spalte
09C6:  FE 50        CP    $50           Spalte 80 erreicht?
09C8:  D0           RET   NC            Ja, dann Ende (Fehler)
09C9:  EB           EX    DE,HL         Zeile/Spalte nach HL
09CA:  22 0B 24     LD    ($240B),HL    Setze Zeile/Spalte

************************************ Neue Cursorposition

09CD:  2A 0B 24     LD    HL,($240B)    Hole Zeile/Spalte
09D0:  CD CE 0C     CALL  $0CCE         Neue Cursoradresse berechnen
09D3:  11 00 14     LD    DE,$1400      Offset für 80-Zeichen-Simulator
09D6:  19           ADD   HL,DE         hinzuaddieren
09D7:  22 09 24     LD    ($2409),HL    und angepaßte Adresse merken
09DA:  C3 4A 0C     JP    $0C4A         Zeile darstellen

************************************ Zeile erniedrigen

09DD:  3A 0C 24     LD    A,($240C)     Hole Cursorzeile
09E0:  B7           OR    A             Setze Flags
09E1:  C8           RET   Z             Zeile 0! Davor geht nicht
09E2:  3D           DEC   A             sonst erniedrige Zeilenzeiger
09E3:  32 0C 24     LD    ($240C),A     und merke Zeile
09E6:  21 04 24     LD    HL,$2404      Setze Bit 6 an $2404
09E9:  CB F6        SET   6,(HL)        als OK-Zeichen
09EB:  18 E0        JR    $09CD         Neue Cursorposition berechnen

************************************ Spalte=0 bzw. Spalte definieren

09ED:  AF           XOR   A             Akku=0 für erste Spalte
09EE:  32 0B 24     LD    ($240B),A     und Spalte definieren
09F1:  3A 0C 24     LD    A,($240C)     Hole Zeile
09F4:  FE 17        CP    $17           letzte Zeile?
09F6:  28 0A        JR    Z,$0A02       Ja, dann Sprung
09F8:  30 03        JR    NC,$09FD      Fehler korrigieren
09FA:  3C           INC   A             Zeile ums eins erhöhen
09FB:  18 E6        JR    $09E3         und merken

09FD:  3E 17        LD    A,$17         Zeile 23 (letzte Zeile)
09FF:  32 0C 24     LD    ($240C),A     merken
```

```
0A02:  21 50 14    LD    HL,$1450     Zweite Zeile Anfangsadresse
0A05:  11 00 14    LD    DE,$1400     Erste Zeile Anfangsadresse
0A08:  01 30 07    LD    BC,$0730     22 Zeilen zu kopieren
0A0B:  ED B0       LDIR               Scrolling
0A0D:  EB          EX    DE,HL        Zieladresse als Quelle
0A0E:  11 31 1B    LD    DE,$1B31     Zweites Zeichen letzte Zeile
0A11:  01 4F 00    LD    BC,$004F     79 Zeichen sind zu füllen
0A14:  CD 24 0B    CALL  $0B24        Fülle letzte Zeile mit Füllz.
0A17:  21 50 1C    LD    HL,$1C50     Anfangsadresse zweite Zeile
                                      (Attribut)
0A1A:  11 00 1C    LD    DE,$1C00     Anfangsadresse erste Zeile
                                      (Attribut)
0A1D:  01 30 07    LD    BC,$0730     22 Zeilen sind zu scrollen
0A20:  ED B0       LDIR               Scrollen im Farbram durchführen
0A22:  EB          EX    DE,HL        1. Zeichen letzte Zeile nach HL
0A23:  11 31 23    LD    DE,$2331     2. Zeichen letzte Zeile
                                      (Farbram)
0A26:  01 4F 00    LD    BC,$004F     79 Zeichen sind zu füllen
0A29:  3A 0D 24    LD    A,($240D)    Farbe für Farbram holen
0A2C:  77          LD    (HL),A       setzen
0A2D:  ED B0       LDIR               und restliche Zeile auch füllen
0A2F:  18 B5       JR    $09E6        OK setzen

************************************** Cursor um eine Stelle nach links

0A31:  3A 0B 24    LD    A,($240B)    Hole Spaltenposition
0A34:  B7          OR    A            Setze Flags
0A35:  C8          RET   Z            Erste Spalte, dann Ende
0A36:  3D          DEC   A            sonst Cursor nach links
0A37:  32 0B 24    LD    ($240B),A    Abspeichern der neuen Spalte
0A3A:  18 91       JR    $09CD        Cursoradresse berechnen

************************************** Cursor um eine Stelle nach rechts

0A3C:  3A 0B 24    LD    A,($240B)    Hole Spalte
0A3F:  3C          INC   A            und um eine Stelle nach rechts
0A40:  FE 50       CP    $50          80. Spalte erreicht?
0A42:  20 F3       JR    NZ,$0A37     Abspeichern neue Position
0A44:  C9          RET                keine Cursorbewegung erfolgt

************************************** Spalte = 0 setzen

0A45:  AF          XOR   A            Akku löschen und
0A46:  18 EF       JR    $0A37        als Spaltenwert abspeichern

************************************** Cursorpos. bis Zeilenende löschen

0A48:  21 CF 0B    LD    HL,$0BCF     Return nach
```

```
0A4B:  E5          PUSH  HL          $0BCF simulieren
0A4C:  CD C2 0C    CALL  $0CC2       Cursorpos. ermitteln und
                                     Restzeichen/Zeile
0A4F:  11 00 14    LD    DE,$1400    Offset für 80-Zeichen-Simulator
0A52:  19          ADD   HL,DE       addieren
0A53:  CD B3 0A    CALL  $0AB3       Füllzeichen setzen
0A56:  79          LD    A,C         Restzeichen/Zeile
0A57:  A7          AND   A           Setze Flags
0A58:  C8          RET   Z           Keine weiteren Zeichen
0A59:  C5          PUSH  BC          Rette Anzahl
0A5A:  E5          PUSH  HL          Rette Quelladresse
0A5B:  54          LD    D,H         Registerpaar DE
0A5C:  5D          LD    E,L         gleich HL
0A5D:  13          INC   DE          plus 1
0A5E:  ED B0       LDIR              Rest bis Zeilenende löschen
0A60:  18 1C       JR    $0A7E       Attribut ebenfalls löschen

************************************ Cursorpos. bis Bildende löschen

0A62:  21 0C 0C    LD    HL,$0C0C    $0C0C als Rücksprungadresse
0A65:  E5          PUSH  HL          auf Stack simulieren
0A66:  11 7F 1B    LD    DE,$1B7F    letzte Spalte in letzter Zeile
0A69:  2A 09 24    LD    HL,($2409)  Cursoradr. 80-Zeichen-Simulator
0A6C:  EB          EX    DE,HL       nach DE, $1B7F nach HL
0A6D:  AF          XOR   A           Carry für Subtraktion löschen
0A6E:  ED 52       SBC   HL,DE       Ermittle Anzahl bis Bildende
0A70:  F8          RET   M           wenn negativ, dann Fehler
0A71:  EB          EX    DE,HL       sonst Ergebnis nach DE
0A72:  28 3F       JR    Z,$0AB3     nur ein Zeichen, dann füllen
0A74:  42          LD    B,D         sonst Anzahl
0A75:  4B          LD    C,E         ins Registerpaar BC
0A76:  54          LD    D,H         und Registerpaar DE (Ziel)
0A77:  5D          LD    E,L         gleich Registerpaar HL
0A78:  13          INC   DE          plus eins
0A79:  C5          PUSH  BC          Anzahl auf Stack
0A7A:  E5          PUSH  HL          Quelle auf Stack
0A7B:  CD 24 0B    CALL  $0B24       Fülle Textzeile mit Attribut
0A7E:  01 00 08    LD    BC,$0800    Offset für Farbram
0A81:  E1          POP   HL          Quelladresse zurückholen
0A82:  09          ADD   HL,BC       und Offset addieren
0A83:  C1          POP   BC          Hole Anzahl Zeichen
0A84:  54          LD    D,H         Zieladresse gleich
0A85:  5D          LD    E,L         Quelladresse
0A86:  13          INC   DE          plus eins
0A87:  3A 0D 24    LD    A,($240D)   Hole Farbe für Farbram VIC
0A8A:  77          LD    (HL),A      und Farbe setzen
0A8B:  ED B0       LDIR              Rest bis Bildende füllen
```

```
0A8D:  C9          RET                Ende der Routine

************************************ an Cursorpos. 1 Stelle einfügen

0A8E:  21 CF 0B    LD    HL,$0BCF     aus RAM in Bildschirm kopieren
0A91:  E5          PUSH  HL           als Rücksprungadresse auf Stack
0A92:  CD C2 0C    CALL  $0CC2        Cursorpos. und Restzeichen
0A95:  21 4F 14    LD    HL,$144F     Letztes Zeichen erste Zeile
0A98:  19          ADD   HL,DE        zur Cursoradresse addieren
0A99:  3D          DEC   A            letzte Spalte?
0A9A:  28 17       JR    Z,$0AB3      Ja, dann überspringe
0A9C:  54          LD    D,H          Sonst Zieladresse gleich
0A9D:  5D          LD    E,L          Quelladresse
0A9E:  2B          DEC   HL           minus eins
0A9F:  C5          PUSH  BC           Rette Anzahl auf Stack
0AA0:  D5          PUSH  DE           Rette Zieladresse auf Stack
0AA1:  ED B8       LDDR               Zeichen hinter Cursor nach
                                      rechts verschieben
0AA3:  EB          EX    DE,HL        HL:=Cursorpos
0AA4:  CD B3 0A    CALL  $0AB3        Hole Füllzeichen
0AA7:  E1          POP   HL           Hole Zieladresse zurück
0AA8:  01 00 08    LD    BC,$0800     Offset für Farbram
0AAB:  09          ADD   HL,BC        zur Quelladresse hinzuaddieren
0AAC:  C1          POP   BC           Anzahl zurückholen
0AAD:  54          LD    D,H          Zieladresse gleich
0AAE:  5D          LD    E,L          Quelladresse
0AAF:  2B          DEC   HL           minus eins
0AB0:  ED B8       LDDR               Farbram ebenfalls verschieben
0AB2:  C9          RET                Ende der Routine

************************************ ($2410) + $20 -> (HL);
                                     Füllzeichen setzen

0AB3:  3A 10 24    LD    A,($2410)    Adresse $2410 auslesen
                                      (Füllzeichen)
0AB6:  C6 20       ADD   A,$20        $20=32 hinzuaddieren
0AB8:  77          LD    (HL),A       und in (HL) ablegen
0AB9:  C9          RET                RETurn aus Routine

************************************ 1 Zeichen an Cursorpos. löschen

0ABA:  21 CF 0B    LD    HL,$0BCF     $0BCF als Rücksprung auf Stack
0ABD:  E5          PUSH  HL           zusätzlich einfügen
0ABE:  CD C2 0C    CALL  $0CC2        Hole Cursoradresse/Restzeichen
0AC1:  11 00 14    LD    DE,$1400     Offset für 80-Zeichen-Simulator
0AC4:  19          ADD   HL,DE        zur Cursoradresse hinzuaddieren
0AC5:  3D          DEC   A            Teste Anzahl/Zeichen
0AC6:  28 EB       JR    Z,$0AB3      Wenn null, dann überspringe
```

```
0AC8:  54          LD    D,H          sonst Zieladresse gleich
0AC9:  5D          LD    E,L          Quelladresse
0ACA:  C5          PUSH  BC           Rette Anzahl
0ACB:  E5          PUSH  HL           Rette Quelladresse
0ACC:  23          INC   HL           Quelladresse=Quelladresse+1
0ACD:  ED B0       LDIR               Zeichen an Cursorpos. löschen
0ACF:  EB          EX    DE,HL        Cursoradresse nach HL
0AD0:  CD B3 0A    CALL  $0AB3        Füllzeichen setzen
0AD3:  E1          POP   HL           Hole Quelladresse
0AD4:  01 01 08    LD    BC,$0801     Offset für Farbram+1
0AD7:  09          ADD   HL,BC        addieren
0AD8:  C1          POP   BC           Hole Zeichenzahl von Stack
0AD9:  54          LD    D,H          Zieladresse gleich
0ADA:  5D          LD    E,L          Quelladresse
0ADB:  23          INC   HL           plus eins
0ADC:  ED B0       LDIR               Farbram ebenfalls verschieben
0ADE:  C9          RET                Ende der Routine

************************************** An Cursorpos. 1 Zeile einfügen

0ADF:  21 0C 0C    LD    HL,$0C0C     Rücksprungsadresse $0C0C
0AE2:  E5          PUSH  HL           auf Stapel einfügen
0AE3:  3A 0C 24    LD    A,($240C)    Hole Cursorzeile
0AE6:  FE 17       CP    $17          Zeile 23 (letzte)?
0AE8:  28 31       JR    Z,$0B1B      Ja, dann nur löschen
0AEA:  D0          RET   NC           Fehler, dann Ende
0AEB:  CD C2 0C    CALL  $0CC2        Hole Cursoradresse/Restzeichen
0AEE:  21 00 14    LD    HL,$1400     Offset für 80-Zeichen-Simulator
0AF1:  19          ADD   HL,DE        hinzuaddieren
0AF2:  E5          PUSH  HL           Rette Quelladresse auf Stapel
0AF3:  11 50 00    LD    DE,$0050     Offset für Anfang nächste Zeile
0AF6:  19          ADD   HL,DE        zur Quelladresse hibzuaddieren
0AF7:  EB          EX    DE,HL        Ergebnis nach (DE)
0AF8:  21 80 1B    LD    HL,$1B80     Adresse letzte Zeile
0AFB:  AF          XOR   A            Carry löschen für Subtraktion
0AFC:  ED 52       SBC   HL,DE        Anzahl Zeichen bis Bildende
0AFE:  44          LD    B,H          Anzahl kommt von
0AFF:  4D          LD    C,L          HL nach BC
0B00:  21 2F 1B    LD    HL,$1B2F     letzte Spalte vorletzte Zeile
0B03:  11 7F 1B    LD    DE,$1B7F     letzte Spalte letzte Zeile
0B06:  C5          PUSH  BC           Rette Anzahl auf Stapel
0B07:  ED B8       LDDR               Zeile an Cursorzeile einfügen
0B09:  C1          POP   BC           Hole ANzahl zurück
0B0A:  21 2F 23    LD    HL,$232F     Adresse Farbram
0B0D:  11 7F 23    LD    DE,$237F     Adresse Farbram (s.o.)
0B10:  ED B8       LDDR               ebenfalls verschieben
0B12:  E1          POP   HL           Hole Quelladresse
0B13:  54          LD    D,H          DE gleich
```

```
0B14:  5D         LD    E,L           Quelladresse
0B15:  13         INC   DE            plus eins
0B16:  01 4F 00   LD    BC,$004F      79 Zeichen
0B19:  18 09      JR    $0B24         Lösche neue Zeile

0B1B:  21 30 1B   LD    HL,$1B30      lediglich die letzte
0B1E:  11 31 1B   LD    DE,$1B31      Zeile löschen,
0B21:  01 4F 00   LD    BC,$004F      da Cursor in letzter Zeile ist
0B24:  3A 10 24   LD    A,($2410)     Hole Füllzeichen
0B27:  C6 20      ADD   A,$20         addiere 32 (Leerzeichen)
0B29:  77         LD    (HL),A        und Füllzeichen setzen
0B2A:  ED B0      LDIR                Rest mit Füllzeichen füllen
0B2C:  C9         RET                 und Ende der Routine

************************************ Lösche Cursorzeile inkl.
                                     Bildschirmverschieben

0B2D:  21 0C 0C   LD    HL,$0C0C      Rücksprungsadresse $0C0C
0B30:  E5         PUSH  HL            auf Stapel hinzufügen
0B31:  3A 0C 24   LD    A,($240C)     Hole Cursorzeile
0B34:  FE 17      CP    $17           Letzte Zeile?
0B36:  28 E3      JR    Z,$0B1B       lediglich letzte Zeile löschen
0B38:  D0         RET   NC            Bei NC Fehler und Ende
0B39:  CD C2 0C   CALL  $0CC2         Cursoradresse/Restzeichen
0B3C:  21 00 14   LD    HL,$1400      Offset für 80-Zeichen-Simulator
0B3F:  19         ADD   HL,DE         zur Cursoradresse addieren
0B40:  E5         PUSH  HL            Rette Quelladresse auf Stapel
0B41:  11 50 00   LD    DE,$0050      Offset für Start Folgezeile
0B44:  19         ADD   HL,DE         addieren
0B45:  EB         EX    DE,HL         Ergebnis nach DE
0B46:  21 80 1B   LD    HL,$1B80      Adresse Statuszeile
0B49:  AF         XOR   A             Lösche Carry für Subtraktion
0B4A:  ED 52      SBC   HL,DE         Ermittelt Zeichen bis Bildende
0B4C:  44         LD    B,H           Anzahl
0B4D:  4D         LD    C,L           nach BC
0B4E:  EB         EX    DE,HL         Startadresse->HL
0B4F:  D1         POP   DE            Hole Startadresse Cursorzeile
0B50:  C5         PUSH  BC            Rette Anzahl auf Stapel
0B51:  E5         PUSH  HL            Rette Quelladresse auf Stapel
0B52:  D5         PUSH  DE            Rette Zieladresse auf Stack
0B53:  ED B0      LDIR                Und Zeile löschen
0B55:  01 00 08   LD    BC,$0800      Offset für Farbram
0B58:  E1         POP   HL            zur Quelladresse
0B59:  09         ADD   HL,BC         hinzuaddieren
0B5A:  EB         EX    DE,HL         Ergebnis nach DE
0B5B:  E1         POP   HL            Hole Startadresse vorletzte
                                      Zeile Farbram
0B5C:  09         ADD   HL,BC         ebenfalls Offset addieren
```

```
0B5D:  C1          POP   BC          Hole Anzahl
0B5E:  ED B0       LDIR              und Farbram auch verschieben
0B60:  18 B9       JR    $0B1B       letzte Zeile löschen

*********************************** <B>:auszuschaltende,
                                    <C>:einzuschaltende Bits
                                    bei Zeichenfarbe

0B62:  78          LD    A,B         auszuschaltende Bits nach <A>
0B63:  E6 70       AND   $70         Bit 7 und Bits 0-3 löschen
0B65:  47          LD    B,A         Ergebnis nach <B>
0B66:  79          LD    A,C         Hole zu setzende Bits
0B67:  E6 70       AND   $70         Bits 0,1,2,3,4,7 löschen
0B69:  4F          LD    C,A         Ergebnis nach C
0B6A:  3A 0D 24    LD    A,($240D)   Hole Attribut
0B6D:  2F          CPL               komplementieren und
0B6E:  B0          OR    B           zu löschende Attribute setzen
0B6F:  2F          CPL               erneut komplementieren und
0B70:  B1          OR    C           zu setzende Eigenschaften setzen
0B71:  32 0D 24    LD    ($240D),A   neues Attribut merken
0B74:  17          RLA               6. Bit ins 7. shiften
0B75:  E6 80       AND   $80         (Reverszeichen) und ausmaskieren
0B77:  32 10 24    LD    ($2410),A   $00 oder $80 als Füllzeichen
0B7A:  C9          RET               Ende der Routine

***********************************

0B7B:  78          LD    A,B         auszugebendes Zeichen <Akku>
0B7C:  D6 20       SUB   $20         Minus ASCII 32
0B7E:  FE 30       CP    $30         kleiner als 48?
0B80:  38 0E       JR    C,$0B90     Ja, dann Sprung
0B82:  0E 30       LD    C,$30       sonst merke 48 als abgezogen
0B84:  CD E5 0C    CALL  $0CE5       weitere Dekodierung
0B87:  D8          RET   C           Alles klar
0B88:  7E          LD    A,(HL)      Hole Farbe
0B89:  0F          RRCA              /2
0B8A:  0F          RRCA              /2=/4
0B8B:  0F          RRCA              /2=/8
0B8C:  0F          RRCA              /2=/16
0B8D:  E6 0F       AND   $0F         Bits 4-7 ausmaskieren
0B8F:  80          ADD   A,B         und <B> wieder hinzuaddieren
0B90:  FE 10       CP    $10         Übertrag ins höherw. Nibble?
0B92:  38 29       JR    C,$0BBD     Nein, neue Farbe definieren
0B94:  FE 20       CP    $20         Rahmen- oder Hintergrundfarbe?
0B96:  38 0B       JR    C,$0BA3     Hintergrundfarbe definieren
```

```
************************************ Rahmenfarbe setzen

0B98:  E6 0F       AND   $0F          Bits 7-4 ausmaskieren
0B9A:  32 0F 24    LD    ($240F),A    und Rahmenfarbe merken
0B9D:  01 20 D0    LD    BC,$D020     Adresse für Rahmenfarbe
0BA0:  ED 79       OUT   (C),A        Rahmenfarbe an VIC übergeben
0BA2:  C9          RET                Ende der Routine

************************************ Hintergrundfarbe 40-Zeichen setzen

0BA3:  E6 0F       AND   $0F          Bits 4-7 ausmaskieren
0BA5:  32 0E 24    LD    ($240E),A    und Hintergrundfarbe merken
0BA8:  01 21 D0    LD    BC,$D021     Adresse für Hintergrundfarbe
0BAB:  ED 79       OUT   (C),A        Hintergrundfarbe an VIC
0BAD:  C9          RET                Ende der Routine

************************************ Hole: <B>:Hintergrund, <C>:Rahmen,
                                     <D>:Zeichenfarbe

0BAE:  3A 0E 24    LD    A,($240E)    Hole Hintergrundfarbe
0BB1:  47          LD    B,A          in <B> merken
0BB2:  3A 0F 24    LD    A,($240F)    Hole Rahm
0BB5:  4F          LD    C,A          in <C> merken
0BB6:  3A 0D 24    LD    A,($240D)    Hole Zeichenfarbe
0BB9:  57          LD    D,A          in <D> merken
0BBA:  E6 0F       AND   $0F          unnötige Bits 7-4 ausmaskieren
0BBC:  C9          RET                Ende der Routine

************************************ neue Farbe definieren

0BBD:  47          LD    B,A          Code nach <B>
0BBE:  3A 0D 24    LD    A,($240D)    Hole alte Farbe
0BC1:  E6 F0       AND   $F0          Bits 0 bis 3 ausmaskieren
0BC3:  B0          OR    B            und neue Farbe reinORen
0BC4:  32 0D 24    LD    ($240D),A    Neue Farbe abspeichern
0BC7:  2A 09 24    LD    HL,($2409)   Hole Cursoradresse
0BCA:  11 00 08    LD    DE,$0800     Offset für Attribut-RAM
0BCD:  19          ADD   HL,DE        addieren
0BCE:  77          LD    (HL),A       neue Farbe setzen

************************************ Zeile aus RAM in Bildschirm

0BCF:  3A 04 24    LD    A,($2404)    Merker für Ausgabe
0BD2:  47          LD    B,A          nach <B>
0BD3:  B7          OR    A            Setze Flags
0BD4:  FC 3C 0C    CALL  M,$0C3C      Bit 7 gesetzt, dann anpassen
0BD7:  3A 02 24    LD    A,($2402)    Hole angepaßte Spalte
0BDA:  B8          CP    B            gleich mit 80-Zeichen-Spalte?
```

```
0BDB:  32 04 24     LD     ($2404),A     merke Spalte
0BDE:  20 2C        JR     NZ,$0C0C      ungleich, dann Sprung
0BE0:  CD C2 0C     CALL   $0CC2         Cursorposition berechnen
0BE3:  21 00 14     LD     HL,$1400      Offset $1400
0BE6:  19           ADD    HL,DE         addieren
0BE7:  EB           EX     DE,HL         und in DE merken
0BE8:  2A 02 24     LD     HL,($2402)    plus der angepaßten Spalte
0BEB:  19           ADD    HL,DE         ergibt Zieladresse
0BEC:  E5           PUSH   HL            auf Stack sichern
0BED:  3A 0C 24     LD     A,($240C)     Hole Zeile
0BF0:  6F           LD     L,A           und nach <L>
0BF1:  CD 70 0C     CALL   $0C70         Zeilenbeginn für 40-Zeichen
0BF4:  EB           EX     DE,HL         und in DE merken
0BF5:  E1           POP    HL            angepaßte Adresse holen
0BF6:  E5           PUSH   HL            und wieder retten
0BF7:  D5           PUSH   DE            rette Zeilenbeginn
0BF8:  3E 01        LD     A,$01         1 Zeile ist zu kopieren
0BFA:  CD 27 0C     CALL   $0C27         kopiere ($1400+SP) in Bildschirm
0BFD:  E1           POP    HL            40-Zeichen-Adresse zurück
0BFE:  01 00 E4     LD     BC,$E400      Offset für Farbram

0C01:  09           ADD    HL,BC         Offset addieren
0C02:  EB           EX     DE,HL         und in DE merken
0C03:  E1           POP    HL            hole Quelladresse
0C04:  01 00 08     LD     BC,$0800      Offset für Bildschirmfarbram
0C07:  09           ADD    HL,BC         zu Quelladresse addieren
0C08:  3E 01        LD     A,$01         1 Zeile
0C0A:  18 1B        JR     $0C27         und kopieren

************************************ 40-Zeichen-Bildschirm kopieren

0C0C:  2A 02 24     LD     HL,($2402)    Hole angepaßte Cursoradresse
0C0F:  3A 08 24     LD     A,($2408)     Anzahl darzustellender
                                         Zeichen/Zeile
0C12:  E5           PUSH   HL            Rette Cursoradresse
0C13:  F5           PUSH   AF            Rette Anzahl/Zeichen
0C14:  11 00 14     LD     DE,$1400      Offset für 80-Zeichen-Simulation
0C17:  19           ADD    HL,DE         addieren
0C18:  11 00 2C     LD     DE,$2C00      tats. Videoram
0C1B:  CD 27 0C     CALL   $0C27         Zeile kopieren
0C1E:  F1           POP    AF            Hole Anzahl
0C1F:  E1           POP    HL            Hole Quelladresse
0C20:  11 00 1C     LD     DE,$1C00      Offset für Farbram
0C23:  19           ADD    HL,DE         addieren
0C24:  11 00 10     LD     DE,$1000      Farbramadresse (angepaßt)
```

```
************************************ (HL) -> (DE) 40 Zeichen in
                                     Bildschirm <A> Zeilen

0C27:  32 03 FF    LD     ($FF03),A    PCRC als Konfigurationsbyte
0C2A:  01 28 00    LD     BC,$0028     40 Zeichen sind zu kopieren
0C2D:  ED B0       LDIR                (HL) -> (DE)
0C2F:  D5          PUSH   DE           rette Zieladresse
0C30:  11 28 00    LD     DE,$0028     40 Zeichen sind kopiert worden
0C33:  19          ADD    HL,DE        addiere zu Quelladresse
0C34:  D1          POP    DE           hole Zieladresse
0C35:  3D          DEC    A            erniedrige den Zähler
0C36:  20 F2       JR     NZ,$0C2A     noch eine Zeile zu kopieren
0C38:  32 01 FF    LD     ($FF01),A    sonst PCRA als Konfig.-Byte
0C3B:  C9          RET                 Ende der Routine

************************************ Spalte anpassen

0C3C:  3A 0B 24    LD     A,($240B)    Hole Spalte
0C3F:  D6 20       SUB    $20          minus 32
0C41:  30 01       JR     NC,$0C44     Kein Übertrag entstanden
0C43:  AF          XOR    A            sonst lösche Akku (=0)
0C44:  E6 F8       AND    $F8          Bits 0-2 ausmaskieren
0C46:  32 02 24    LD     ($2402),A    angepaßte Spalte merken
0C49:  C9          RET                 Ende der Routine

************************************ Zeile kopieren

0C4A:  CD 69 0C    CALL   $0C69        $2406 löschen
0C4D:  CD CF 0B    CALL   $0BCF        Zeile kopieren
0C50:  3A 02 24    LD     A,($2402)    Hole angepaßte Spalte
0C53:  47          LD     B,A          in <B> merken
0C54:  2A 0B 24    LD     HL,($240B)   Hole Zeile/Spalte
0C57:  7D          LD     A,L          Spalte nach <A>
0C58:  90          SUB    B            minus angepaßter Spalte
0C59:  38 0E       JR     C,$0C69      zu klein, dann $2406 löschen
0C5B:  FE 28       CP     $28          zu groß - größer als 40?
0C5D:  30 0A       JR     NC,$0C69     dann $2406 löschen
0C5F:  4F          LD     C,A          Spalte nach <C>
0C60:  06 00       LD     B,$00        Hi-Byte von BC löschen
0C62:  6C          LD     L,H          <L>=Zeile
0C63:  CD 70 0C    CALL   $0C70        Und Startadresse Zeile berechnen
0C66:  09          ADD    HL,BC        addiere Spalte
0C67:  18 03       JR     $0C6C        und merken der Adresse
0C69:  21 00 00    LD     HL,$0000     Startadresse ist erste Position
0C6C:  22 06 24    LD     ($2406),HL   merken der Position
0C6F:  C9          RET                 Ende der Routine
```

```
************************************ Zeile*40 + Offset

0C70:  26 00       LD    H,$00       Hi-Byte löschen
0C72:  29          ADD   HL,HL       *2
0C73:  29          ADD   HL,HL       *2
0C74:  29          ADD   HL,HL       *2=*8
0C75:  54          LD    D,H         nach DE
0C76:  5D          LD    E,L         merken
0C77:  29          ADD   HL,HL       *2
0C78:  29          ADD   HL,HL       *2=*32
0C79:  19          ADD   HL,DE       *32+*8 ergibt *40
0C7A:  11 00 2C    LD    DE,$2C00    Offset von $2C00 (Textanfang)
0C7D:  19          ADD   HL,DE       addieren
0C7E:  C9          RET               Ende der Routine

************************************ ASCII-Code-Anpassung für 40-Zeichen

0C7F:  78          LD    A,B         Zeichen nach <Akku>
0C80:  FE 40       CP    $40         Ist es ASCII 64 (@)
0C82:  28 3C       JR    Z,$0CC0     Ja, dann Poke-Code=0
0C84:  D8          RET   C           Bei kleiner, Ende
0C85:  FE 5B       CP    $5B         kleiner als ASCII "Z"+1?
0C87:  D8          RET   C           Ja, dann Rückkehr
0C88:  D6 40       SUB   $40         64 abziehen
0C8A:  FE 20       CP    $20         Ist es Apostroph?
0C8C:  28 19       JR    Z,$0CA7     Ja, dann Codiere
0C8E:  38 23       JR    C,$0CB3     Kleiner als Apostroph
0C90:  D6 20       SUB   $20         minus ASCII 32
0C92:  FE 1B       CP    $1B         Grenze kleine Buchstaben
0C94:  D8          RET   C           Kleiner, dann Ende
0C95:  FE 1B       CP    $1B         erneut vergleichen
0C97:  28 11       JR    Z,$0CAA     kleiner ASCII "ä"?
0C99:  FE 1C       CP    $1C         kleines "ö"?
0C9B:  28 10       JR    Z,$0CAD     Ja, dann codiere
0C9D:  FE 1D       CP    $1D         kleines "ü"?
0C9F:  28 0F       JR    Z,$0CB0     Ja, dann codiere
0CA1:  FE 1E       CP    $1E         Ist es "ß"?
0CA3:  C0          RET   NZ          Nein, dann Rückkehr
0CA4:  3E 40       LD    A,$40       Sonst codiere 64
0CA6:  C9          RET               und Ende der Routine

************************************ ASCII-Code 126 generieren

0CA7:  3E 7E       LD    A,$7E       126
0CA9:  C9          RET               Ende der Routine
```

```
************************************ ASCII-Code 115

0CAA:  3E 73        LD    A,$73         115
0CAC:  C9           RET                 Ende der Routine

************************************ ASCII-Code 93

0CAD:  3E 5D        LD    A,$5D         93
0CAF:  C9           RET                 Ende der Routine

************************************ ASCII-Code 107

0CB0:  3E 6B        LD    A,$6B         107
0CB2:  C9           RET                 Ende der Routine

************************************ ASCII-Code 28

0CB3:  FE 1C        CP    $1C           ASCII 28?
0CB5:  28 06        JR    Z,$0CBD       Ja, dann zuordnen
0CB7:  FE 1F        CP    $1F           ASCII 31?
0CB9:  C0           RET   NZ            Nein, dann Ende mit Flag
0CBA:  3E 64        LD    A,$64         Sonst ASCII 100
0CBC:  C9           RET                 Ende der Routine

************************************ ASCII-Code 127 zuordnen

0CBD:  3E 7F        LD    A,$7F         127
0CBF:  C9           RET                 Ende der Routine

************************************ <Akku> löschen

0CC0:  AF           XOR   A             <Akku> löschen
0CC1:  C9           RET                 Ende der Routine

************************************ <C>:79-Spalte und Cursorpos.
                                     berechnen

0CC2:  2A 0B 24     LD    HL,($240B)    Hole Zeile/Spalte
0CC5:  18 03        JR    $0CCA         weitermachen

0CC7:  2A 13 24     LD    HL,($2413)    Hole Zeile/Spalte
0CCA:  3E 4F        LD    A,$4F         dezimal 79
0CCC:  95           SUB   L             minus Spalte
0CCD:  4F             LD    C,A             in <C> merken (Restanzahl bis
80)
```

```
************************************ Cursorposition berechnen
                                     <H>:Zeile, <L>:Spalte

0CCE:  45          LD    B,L          <B> gleich Spalte
0CCF:  6C          LD    L,H          <L> ist nun Zeile
0CD0:  26 00       LD    H,$00        lösche Hi-Byte
0CD2:  29          ADD   HL,HL        *2
0CD3:  29          ADD   HL,HL        *2
0CD4:  29          ADD   HL,HL        *2
0CD5:  29          ADD   HL,HL        *2=*16
0CD6:  54          LD    D,H          <DE> wird nun mit
0CD7:  5D          LD    E,L          Zeile mal 16 belegt
0CD8:  29          ADD   HL,HL        *2
0CD9:  29          ADD   HL,HL        *2 ergibt *64
0CDA:  19          ADD   HL,DE        plus *16 ergibt *80
0CDB:  EB          EX    DE,HL        Zeile mal 80 nach <DE>
0CDC:  68          LD    L,B          Spalte nach <L>
0CDD:  26 00       LD    H,$00        Hi-Byte löschen
0CDF:  19          ADD   HL,DE        und Zeile*80 addieren
0CE0:  06 00       LD    B,$00        Lösche Hi-Byte von <BC>
0CE2:  3C          INC   A            <A>:=Anzahl möglicher Zeichen
0CE3:  C9          RET                Ende der Routine

************************************ ASCII-Dekodierung Cont'd

0CE4:  78          LD    A,B          Zeichen nach <Akku>
0CE5:  2A 0D FD    LD    HL,($FD0D)   Tabellenzeiger
0CE8:  D6 30       SUB   $30          Minus ASCII 48 "0"
0CEA:  D8          RET   C            Kleiner, dann Ende
0CEB:  B9          CP    C            Sonst vergleiche mit <C>
                                      (abgezogener Wert)
0CEC:  3F          CCF                Carry-Flag negieren
0CED:  D8          RET   C            Und bei größer/gleich Ende
0CEE:  47          LD    B,A          Sonst Zeichen nach <B>
0CEF:  E6 0F       AND   $0F          Bits 0-3 ausmaskieren
0CF1:  5F          LD    E,A          Lo-Byte gleich <Akku>
0CF2:  16 00       LD    D,$00        und Hi-Byte löschen
0CF4:  19          ADD   HL,DE        zur Tabellenbasis addieren
0CF5:  78          LD    A,B          Zeichen wieder nach <Akku>
0CF6:  E6 30       AND   $30          Bits 6,7 und 0-3 ausmaskieren
0CF8:  47          LD    B,A          Zeichen wieder nach <B>
0CF9:  C9          RET                Ende der Routine

************************************ Ton erklingen lassen

0CFA:  01 18 D4    LD    BC,$D418     SID Register 24
0CFD:  2A 10 FD    LD    HL,($FD10)   Hole Attack/Decay/Volume
0D00:  ED 61       OUT   (C),H        Gesamtlautstärke/Filter
```

```
0D02:  0E 05      LD   C,$05        Register 5 des SID: Attack/Decay
0D04:  ED 69      OUT  (C),L        definieren
0D06:  2A 12 FD   LD   HL,($FD12)   Hole Sustain/Release/Frequenz
0D09:  0C         INC  C            Register 6 des SID
0D0A:  ED 61      OUT  (C),H        Sustain/Release definieren
0D0C:  0E 01      LD   C,$01        Register 1 des SID: Frequenz
0D0E:  ED 69      OUT  (C),L        Frequenz (HI) definieren
0D10:  2A 14 FD   LD   HL,($FD14)   Hole Ein/Ausschalten
0D13:  0E 04      LD   C,$04        Register 4 des SID
0D15:  ED 61      OUT  (C),H        Einschalten des Tones
0D17:  ED 69      OUT  (C),L        Bit zum Sustain löschen
0D19:  C9         RET               Ende der Tonerklingungsroutine

************************************ Wird nach $1100 kopiert

0D1A:  1100: A9 00      LDA #$00      Konfigurationsbyte
0D1C:  1102: 8D 00 FF   STA $FF00     setzen (alles ROM)
0D1F:  1105: 6C FC FF   JMP ($FFFC)   Reset des C128-Modus

************************************ Wird nach $3000 kopiert
                                     Lesen eines Diskettenblockes

0D22: 3000: A9 00      LDA #$00     Noch ist kein
0D24: 3002: 8D 06 FD   STA $FD06    Fehler aufgetreten
0D27: 3005: 20 11 30   JSR $3011    Laden des Blockes
0D2A: 3008: 78         SEI          Interrupt verhindern
0D2B: 3009: A9 3E      LDA #$3E     RAM und System I/O
0D2D: 300B: 8D 00 FF   STA $FF00    als Konfigurationsbyte
0D30: 300E: 4C D0 FF   JMP $FFD0    und Z-80 einschalten

0D33: 3011: D8         CLD          Lösche Dezimalflag
0D34: 3012: AD 01 FD   LDA $FD01    Flag für Vektoren setzen
0D37: 3015: D0 21      BNE $3038    gelöscht?
0D39: 3017: A2 00      LDX #$00     ROM und System I/O
0D3B: 3019: 8E 00 FF   STX $FF00    einschalten
0D3E: 301C: 8E 1A D0   STX $D01A    IMR im VIC-Chip löschen
0D41: 301F: A2 63      LDX #$63     Lo-Byte anzuspringender Routine
0D43: 3021: A0 31      LDY #$31     und Hi-Byte derselben
0D44: 3023: 8E 14 03   STX $0314    Lo-Byte als IRQ-Routine
0D48: 3026: 8C 15 03   STY $0315    Hi-Byte als IRQ-Routine
0D4B: 3029: 8E 16 03   STX $0316    Lo-Byte als BRK-Routine
0D4E: 302C: 8C 17 03   STY $0317    Hi-Byte als BRK-Routine
0D51: 302F: 8E 18 03   STX $0318    Lo-Byte als NMI-Routine
0D54: 3032: 8C 19 03   STY $0319    Hi-Byte als NMI-Routine
0D57: 3035: 4C D7 30   JMP $30D7    Weitermachen bei $30D7
```

```
********************************** Lesen von Block (Spur/Sektor)
                                   & Ablage im Speicher

0D5A: 3038: AD 18 FD      LDA $FD18      Lo-Byte Zieladresse
0D5D: 303B: 85 20         STA $20        nach $20
0D5F: 303D: AD 19 FD      LDA $FD19      Hi-Byte Zieladresse
0D62: 3040: 85 21         STA $21        nach $21 kopieren
0D64: 3042: AD 03 FD      LDA $FD03      Hole Spur
0D67: 3045: 8D BF 31      STA $31BF      in Diskettenkommando kopieren
0D6A: 3048: 20 78 31      JSR $3178      macht aus <Akku> ASCII "xx"
0D6D: 304B: 8E B1 31      STX $31B1      Zehnerstelle Track und
0D70: 304E: 8D B0 31      STA $31B0      Einerstelle in Kommandozeile
0D73: 3051: AD 04 FD      LDA $FD04      Hole Sektornummer
0D76: 3054: 8D BE 31      STA $31BE      Sektornummer für FSD übergeben
0D79: 3057: 20 78 31      JSR $3178      und in ASCII wandeln
0D7C: 305A: 8E AE 31      STX $31AE      Zehnerstelle Sektor sowie
0D7F: 305D: 8D AD 31      STA $31AD      Einerstelle in Kommandozeile
0D82: 3060: AD 08 FD      LDA $FD08      Teste, ob Kanal eröffnet wurde
0D85: 3063: D0 2B         BNE $3090      Nein, dann Sprung nach $3090
0D87: 3065: 8D 00 FF      STA $FF00      Konfig.-Byte auf 0F (ROM) setzen
0D8A: 3068: A2 0B         LDX #$0B       Kanal 11 (#) als Eingabe-
0D8C: 306A: 20 C6 FF      JSR $FFC6      kanal definieren
0D8F: 306D: B0 16         BCS $30A7      Bei Fehler nach $30A7
0D91: 306F: 20 CC FF      JSR $FFCC      CLRCH; Ein/Ausgabe wieder normal
0D94: 3072: 20 31 31      JSR $3131      Track/Sektor lesen
0D97: 3075: 20 99 31      JSR $3199      Kanal 11 (#) als Eingabekanal
0D9A: 3078: A0 00         LDY #$00       Y-Index auf null
0D9C: 307A: 20 CF FF      JSR $FFCF      BASIN; Zeichen von Floppy holen
0D9F: 307D: 91 20         STA ($20),Y    und Zeichen im RAM ablegen
0DA1: 307F: C8            INY            nächstes Byte
0DA2: 3080: D0 F8         BNE $307A      Ende noch nicht erreicht
0DA4: 3082: 4C CC FF      JMP $FFCC      CLRCH; Ein/Ausgabe wieder normal

0DA7: 3085: A9 FF         LDA #$FF       Block konnte nicht geholt werden
0DA9: 3087: 2C            .Byte $2C      Skip; Überspringe Folgekommando
0DAA: 3088: A9 0D         LDA #$0D       Kennzeichen für Fehler
0DAC: 308A: 8D 06 FD      STA $FD06      aufgetreten setzen
0DAF: 308D: 4C 08 30      JMP $3008      wieder in Z-80-Teil

********************************** Daten von Floppy lesen

0DB2: 3090: A9 00         LDA #$00       ROM und System I/O als
0DB4: 3092: 8D 00 FF      STA $FF00      Konfigurationsbyte setzen
0DB7: 3095: A2 0F         LDX #$0F       Logische Filenummer 15
0DB9: 3097: 20 C9 FF      JSR $FFC9      Befehlskanal als Ausgabekanal
0DBC: 309A: B0 EC         BCS $3088      bei Fehler anzuspringen
0DBE: 309C: A0 06         LDY #$06       Sechs Zeichen auszugeben
0DC0: 309E: B9 BC 31      LDA $31BC,Y    Hole Zeichen aus Tabelle
```

```
0DC3: 30A1: 20 D2 FF     JSR $FFD2      Ausgabe Zeichen auf Befehlskanal
0DC6: 30A4: 88           DEY            Nächstes Zeichen
0DC7: 30A5: D0 F7        BNE $309E      Noch weitere Zeichen auszugeben
0DC9: 30A7: 20 CC FF     JSR $FFCC      CLRCH; Ein/Ausgabe wieder normal
0DCC: 30AA: 2C 0D DC     BIT $DC0D      Teste ICR
0DCF: 30AD: AE BD 31     LDX $31BD      Anzahl der Datenblöcke
0DD2: 30B0: 20 4F 31     JSR $314F      Hole Datenbyte (Fast-Modus)
0DD5: 30B3: 29 0E        AND #$0E       Teste Bits 1-3
0DD7: 30B5: D0 D1        BNE $3088      Fehler aufgetreten
0DD9: 30B7: A0 00        LDY #$00       Index auf Null
0DDB: 30B9: 20 4F 31     JSR $314F      Hole Datenbyte (Fast-Modus)
0DDE: 30BC: 91 20        STA ($20),Y    Lege Zeichen im RAM ab
0DE0: 30BE: C8           INY            erhöhe den Zeiger
0DE1: 30BF: D0 F8        BNE $30B9      Noch nicht alle Zeichen
0DE3: 30C1: E6 21        INC $21        Erhöhe Hi-Byte des Pointers
0DE5: 30C3: CA           DEX            Erniedrige den Blockzähler
0DE6: 30C4: D0 EA        BNE $30B0      Weitere Blocks
0DE8: 30C6: AD 00 DD     LDA $DD00      PRA CIA2 holen
0DEB: 30C9: 29 EF        AND #$EF       CLK-Bit ausmaskieren
0DED: 30CB: 8D 00 DD     STA $DD00      und wieder zurück
0DF0: 30CE: 60           RTS            Ende der Routine

0DF1: 30CF: A9 0F        LDA #$0F       Logische Filenummer 15
0DF3: 30D1: 8D 06 FD     STA $FD06      in $FD06 ablegen
0DF6: 30D4: 4C 08 30     JMP $3008      Z-80 einschalten

0DF9: 30D7: A9 0F        LDA #$0F       Logische Filenummer
0DFB: 30D9: 18           CLC            Lösche Carry als Flag
0DFC: 30DA: 20 C3 FF     JSR $FFC3      Schließen des Kanals
0DFF: 30DD: A9 0F        LDA #$0F       Logische Filenummer
0E01: 30DF: 8D 08 FD     STA $FD08      Logische Filenummer merken
0E03: 30E2: A2 08        LDX #$08       Geräteadresse
0E06: 30E4: A8           TAY            15 als Sekundäradresse
0E07: 30E5: 20 BA FF     JSR $FFBA      SETLFS; Setzen der log.
                                        Fileparameter
0E0A: 30E8: A9 00        LDA #$00       Schnellmodus der
0E0C: 30EA: 8D 1C 0A     STA $0A1C      Floppy ausschalten
0E0F: 30ED: AA           TAX            Beide Konfigurationsindize
0E10: 30EE: 20 68 FF     JSR $FF68      für SETBNK auf null setzen
0E13: 30F1: A9 04        LDA #$04       Länge Filename
0E15: 30F3: A2 B8        LDX #$B8       Adresse Lo des Filenamens
0E17: 30F5: A0 31        LDY #$31       Adresse Hi des Filenamens
0E19: 30F7: 20 BD FF     JSR $FFBD      SETNAM; Filenamenparameter
0E1C: 30FA: 20 C0 FF     JSR $FFC0      OPEN der Datei
0E1F: 30FD: B0 D0        BCS $30CF      Fehler aufgetreten, dann Sprung
0E21: 30FF: 20 B7 FF     JSR $FFB7      Statusbyte I/O holen
0E24: 3102: 2A           ROL A          Bit 7 ins Carry shiften
0E25: 3103: B0 CA        BCS $30CF      Fehler aufgetreten, dann Sprung
```

```
0E27: 3105: 2C 1C 0A      BIT $0A1C      Teste das Fast-Serial-Bit
0E2A: 3108: 70 26         BVS $3130      ist gesetzt, dann Sprung
0E2C: 310A: A9 0B         LDA #$0B       Logische Filenummer 11
0E2E: 310C: 18            CLC            Lösche Carry als Flag
0E2F: 310D: 20 C3 FF      JSR $FFC3      Schließe Datei 11
0E32: 3110: A9 0B         LDA #$0B       Logische Filenummer 11
0E34: 3112: A2 08         LDX #$08       Gerätenummer 8
0E36: 3114: A0 08         LDY #$08       Sekundäradresse 8
0E38: 3116: 20 BA FF      JSR $FFBA      SETLFS; Filedaten speichern
0E3B: 3119: A9 00         LDA #$00       LFN als nicht geöffnet
0E3D: 311B: 8D 08 FD      STA $FD08      an $FD08 löschen
0E40: 311E: AA            TAX            Beide Konfigurationsindizes
0E41: 311F: 20 68 FF      JSR $FF68      sind null; SETBNK
0E44: 3122: A9 01         LDA #$01       Länge des Filenamens
0E46: 3124: A2 BC         LDX #$BC       Lo-Byte des Filenamens
0E48: 3126: A0 31         LDY #$31       Hi-Byte des Filenamens
0E4A: 3128: 20 BD FF      JSR $FFBD      SETNAM; Adresse Filename setzen
0E4D: 312B: 20 C0 FF      JSR $FFC0      OPEN 11,8,8,"#"; Öffne Datei
0E50: 312E: B0 9F         BCS $30CF      Fehler aufgetreten
0E52: 3130: 60            RTS            sonst Ende der Routine

************************************ Track/Sektor lesen

0E53: 3131: 20 A4 31      JSR $31A4      Befehlskanal auf Ausgabe legen
0E56: 3134: A0 0D         LDY #$0D       13 Zeichen sind auszugeben
                                         "U1:8 0 tt ss<CR>"
0E58: 3136: B9 AB 31      LDA $31AB,Y    Hole Zeichen aus der Tabelle
0E5B: 3139: 20 D2 FF      JSR $FFD2      und ausgeben
0E5E: 313C: 88            DEY            Zähler erniedrigen
0E5F: 313D: D0 F7         BNE $3136      und Sprung, wenn weitere Zeichen
0E61: 313F: 20 CC FF      JSR $FFCC      CLRCH; Ein/Ausgabe wieder normal
0E64: 3142: 20 89 31      JSR $3189      Befehlskanal zum Lesen Öffnen
0E67: 3145: F0 05         BEQ $314C      kein Fehler aufgetreten
0E69: 3147: A9 0D         LDA #$0D       Merker für Fehler
0E6B: 3149: 8D 06 FD      STA $FD06      setzen
0E6E: 314C: 4C CC FF      JMP $FFCC      CLRCH; Ein/Ausgabe wieder normal

************************************ Warte, bis SDR (Serial Data
                                     Register) fertig und
                                     hole dann Datenbyte

0E71: 314F: 78            SEI            Interrupt verhindern
0E72: 3150: AD 00 DD      LDA #$DD00     Hole PRA CIA2
0E75: 3153: 49 10         EOR #$10       Clock-Leitung negieren
0E77: 3155: 8D 00 DD      STA $DD00      Negiert zurückliefern
0E7A: 3158: A9 08         LDA #$08       SDR voll/leer-Bit abtesten
0E7C: 315A: 2C 0D DC      BIT $DC0D      SDR ist noch nicht fertig
0E7F: 315D: F0 FB         BEQ $315A      Warte, bis Timer Unterlauf hat
```

```
0E81: 315F: AD 0C DC      LDA $DC0C     Hole SDR Serial Data Register
0E84: 3162: 60            RTS           Ende der Routine

0E85: 3163: AD 0D DC      LDA $DC0D     Lösche Interruptregister im CIA1
0E88: 3166: AD 0D DD      LDA $DD0D     Lösche Interruptregister im CIA2
0E8B: 3169: A9 0F         LDA #$0F      Lösche Interruptregister
0E8D: 316B: 8D 19 D0      STA $D019     des VIC-Chip
0E90: 316E: 68            PLA           Konfiguration
0E91: 316F: 8D 00 FF      STA $FF00     wiederherstellen
0E94: 3172: 68            PLA           Y-Register
0E95: 3173: A8            TAY           wiederherstellen
0E96: 3174: 68            PLA           X-Register
0E97: 3175: AA            TAX           wiederherstellen
0E98: 3176: 68            PLA           Akku wiederherstellen
0E99: 3177: 40            RTI           Ende der Interrupt-Routine

********************************** macht aus <Akku> zwei ASCII-Codes

0E9A: 3178: D8            CLD           Lösche Dezimalflag
0E9B: 3179: A2 30         LDX #$30      Zehnerstelle zunächst einmal "0"
0E9D: 317B: 38            SEC           Setze Carry-Flag für Subtraktion
0E9E: 317C: E9 0A         SBC #$0A      Zehn subtrahieren (probeweise)
0EA0: 317E: 90 03         BCC $3183     Zuviel subtrahiert
0EA2: 3180: E8            INX           sonst Zehnerstelle erhöhen
0EA3: 3181: B0 F9         BCS $317C     und weitermachen
0EA5: 3183: 69 3A         ADC #$3A      Fehler korrigieren und ASCII-
                                        Basis addieren
0EA7: 3185: 60            RTS           Ende der Routine

************************************ Testet, ob auf Befehlskanal Fehler
                                     vorliegt

0EA8: 3186: 20 D7 30      JSR $30D7     Fehler bei Kanal 15 aufgetreten
0EAB: 3189: A2 0F         LDX #$0F      Befehlskanal zum
0EAD: 318B: 20 C6 FF      JSR $FFC6     Lesen öffnen; CHKIN
0EB0: 318E: B0 F6         BCS $3186     Fehler aufgetreten
0EB2: 3190: 20 CF FF      JSR $FFCF     BASIN; Zeichen holen
0EB5: 3193: C9 30         CMP #$30      Akku "0" vergleichen (dann OK)
0EB7: 3195: 60            RTS           Ende der Routine

************************************ Kanal 11 (#) zum Lesen vorbereiten

0EB8: 3196: 20 0A 31      JSR $310A     Fehler bei Kanal 11 aufgetreten
0EBB: 3199: A2 0B         LDX #$0B      Kanal 11 auf Eingabe
0EBD: 319B: 20 C6 FF      JSR $FFC6     legen; CHKIN
0EC0: 319E: B0 F6         BCS $3196     Fehler aufgetreten
0EC2: 31A0: 60            RTS           Ende der Routine
```

```
************************************ Kanal 15 (Befehlskanal) auf Ausgabe

0EC3: 31A1: 20 D7 30        JSR $30D7      Fehler bei Kanal 15 aufgetreten
0EC6: 31A4: A2 0F           LDX #$0F       Kanal 15 als Ausgabekanal
0EC8: 31A6: 20 C9 FF        JSR $FFC9      durch CKOUT-Routine definieren
0ECB: 31A9: B0 F6           BCS $31A1      Fehler aufgetreten
0ECD: 31AB: 60              RTS            Ende der Routine

0ECE: 31AC: 0D 73 73 20 74 74 20 30       .ss tt 0
0ED6: 31B4: 20 38 3A 31 55 30 4C 00        8:1U0L.
0EDE: 31BC: 23 01 00 00 00 30 55          #....0U

************************************ Wird nach $FFD0 kopiert (8502-Code)

0EE5: ($FFD0) 78              SEI            Interrupt verhindern
0EE6: ($FFD1) A9 3E           LDA #$3E       Konfigurationsbyte $3E setzen
0EE8: ($FFD3) 8D 00 FF        STA $FF00      RAM und System I/O
0EEB: ($FFD6) A9 B0           LDA #$B0       Z-80 einschalten
0EED: ($FFD8) 8D 05 D5        STA $D505      im MCR
0EF0: ($FFDB) EA              NOP            Wartewirkung
0EF1: ($FFDC) 4C 00 30        JMP $3000      Sprung in nächsten Routinenteil
0EF4: ($FFDF) EA              NOP            Buffer

************************************ Wird nach $FFE0 kopiert

0EF5: ($FFE0) F3              DI                   Interrupts unterbinden
0EF6: ($FFE1) 3E 3E           LD    A,$3E          Konfigurationsbyte
0EF8: ($FFE3) 32 00 FF        LD    ($FF00),A setzen
0EFB: ($FFE6) 01 05 D5        LD    BC,$D505  Mode-Config.-Register
0EFE: ($FFE9) 3E B1           LD    A,$B1     und 8502 einschalten
0F00: ($FFEB) ED 79           OUT   (C),A     durch Bit-0-setzen
0F02: ($FFED) 00              NOP             Bufferwirkung
0F03: ($FFEE) CF              RST   $08       Sprung in Z-80-Teil

************************************ Allerlei Tabellen

0F04:  9E FF BD 58 6F CE/00 0F       ab / angepaßte Farbtabelle
0F0C:  08 07 0B 04 02 0D 0A 0C       für 40-Zeichen-Schirm
0F14:  09 06 01 05 03 0E/00 60
0F1C:  30 18 0C 06 03 00 18 3C
0F24:  66 00 00 00 00 00 00 00
0F2C:  00 00 00 00 7F 00 60 30
0F34:  18 00 00 00 00 00 1C 30
0F3C:  30 60 30 30 1C 00 18 18
0F44:  18 18 18 18 18 00 38 0C
0F4C:  0C 06 0C 0C 38 00 00 1B
0F54:  2A 66 00 00 00 00 00 00
0F5D:  00 00 00 41 7F 00 00 F2
```

```
0F64:  5B 39 01 4E 65 37 06 03
0F6C:  1E 07 0B 68 4B 34 17 01
0F74:  44 62 2D 18 12 0B 63 59
0F7C:  31 17 00 0B 59 72 2B 18
0F84:  0F 63 00 4F 2B 05 4C 68
0F8C:  2D 17 16 69 49 25 17 13
0F94:  45 68 29 18 17 07 0C 68
0F9C:  4B 34 13 0F 05 4B 70 31
0FA4:  31 0D 0D 08 08 6C/0D 3F        ab / MMU-Registerbelegungen
0FAC:  7F 3E 7E B0 0B 00 00 01
0FB4:  00/00 05 0A 0F 14 04 09        angepaßte Sektornummern nach
0FBC:  0E 13 03 08 0D 12 02 07        verschiedenen
                                      Sektorgrößenbezirken
0FC4:  0C 11 01 06 0B 10 00 05        aufgeteilt.
0FCC:  0A 0F 01 06 0B 10 02 07        Durch dieses Verfahren wird der
0FD4:  0C 11 03 08 0D 12 04 09        Zugriff auf Diskette zeitlich
0FDC:  0E 00 05 0A 0F 02 07 0C        noch optimiert
0FE4:  11 04 09 0E 01 06 0B 10
0FEC:  03 08 0D 00 05 0A 0F 03
0FF4:  08 0D 01 06 0B 10 04 09
0FFC:  0E 02 07 0C
```

11. Tips und Tricks

Dieses Kapitel kann natürlich nicht das gleichnamige Buch ersetzen, jedoch wollen wir Ihnen die wichtigsten bzw. nützlichsten Dinge erläutern, die wir herausgefunden haben.

Sie können anhand der Beispiele auch ersehen wie man sich die vorliegende Zero-Page bzw. das dokumentierte ROM-Listing zu Nutze machen kann - denn dazu sind die Informationen ja schließlich da.

11.1 STOP-Taste sperren

Es kommt nicht selten vor, da will man dem Benutzer es nicht ermöglichen, durch Betätigen der STOP-Taste das Programm unterbrechen zu können - in manchen Situationen kann es ja gar gefährlich werden, wenn man aus Versehen einmal auf die STOP-Taste kommt.

Will man ein solches Problem lösen, so sieht man in der Zero-Page nach. Hier befindet sich ab Adresse $0300 eine Tabelle mit Sprungkommandos für die wichtigsten Kernal-Routinen. Dieser Bereich stellt praktisch eine Schnittstelle für den Programmierer zum Betriebssystem dar, da er auf bestimmte Abläufe einwirken kann, indem er einfach die Sprungkommandos *verbiegt* (meistens dann auf eine eigene Routine).

An Adresse $0328 befindet sich der Vektor für die Kernal-STOP-Routine - er zeigt auf $F66E. Genau an dieser Adresse $F66E wird der aktuelle Zustand der STOP-Taste aus der Zero-Page-Adresse $91 ermittelt. Diese Adresse $91 wiederum wird immer von der IRQ-Routine auf dem neuesten Stand gehalten. Überspringt man nun diese Abfrage, so erreicht man, daß ein eventuelles Betätigen der STOP-Taste von der STOP-Abfrageroutine nicht erkannt würde. Wir müssen also lediglich die Adresse $0328 korrigieren: Lassen Sie uns in diese Adresse einmal das Lo-Byte des Folgekommandos der STOP-Routine

hineinschreiben. Dies erreichen in BASIC wir durch folgenden POKE:

POKE DEC("0328"),112 : REM STOP-Taste sperren

Der Vektor $0328/$0329 zeigt nun nicht mehr auf $F66E sondern auf $F670. Und siehe da: das Betriebssystem reagiert nicht mehr auf Betätigung der STOP-Taste, weder beim Programmablauf noch beim Listen o.ä.

Damit hätten wir das erzielt, was wir uns Eingangs vorgenommen haben. Jedoch trübt noch ein Fleck unser Wässerchen: Ist jemand so schlau, und betätigt die STOP-Taste und die Restore-Taste gleichzeitig, so wird unser Programm trotzdem unterbrochen! In der NMI-Routine wird zwar auch die STOP-Abfrage-Routine an der Adresse $F66E aufgerufen, allerdings wird nicht über den Vektor an Adresse $0328 gesprungen, so daß das Betätigen der STOP-Taste erkannt werden kann.

11.2 STOP-RESTORE-Kombination sperren

Wird diese Kombination auf der Tastatur betätigt, so wird die sogenannte NMI-Routine aufgerufen. NMI steht für Non-Maskable-Interrupt - es wird also ein Interrupt ausgelöst, den man nicht so einfach mittels dem SEI-Kommando ausschalten kann.

Aber auch hierfür gibt es einen Vektor im Zero-Page-Bereich. Dieser für die NMI-Routine verantwortliche Vektor befindet sich an der Adresse $0318 und zeigt auf die NMI-Routine im Kernal an Adresse $FA40.

Will man nun nicht, daß ein BASIC-Warmstart bei o.g. Tastenkombination vollzogen wird, so muß man den NMI-Vektor auf das Ende der NMI-Routine legen. Es empfiehlt sich hier, den Vektor auf $FA62 umzulegen, da an dieser Stelle in die IRQ-Rücksprungroutine gesprungen wird, in der die Register wieder rückgesetzt werden und ein RTI ausgeführt wird.

Um nun aber die NMI-Routine zu verbiegen, ist folgendes BASIC-Kommando notwendig:

POKE DEC("0318"),98 : NMI verbiegen

Nachdem Sie dieses POKE-Kommando in Ihrem Programm integriert haben, zusammen mit dem Sperren der STOP-Taste, kommt niemand mehr aus Ihrem laufenden Programm heraus, es sei denn, er konstruiert einen RESET am User- oder Expansion-Port, den man aber auch abfangen kann...

11.3 Der IRQ-Vektor

Die IRQ-Routine im Kernal wird ca. jede 1/60-Sekunde aufgerufen. Zuständig hierfür ist der CIA, der durch die Timer diesen Interrupt regelmäßig auslöst.

Der Vektor für die IRQ-Routine befindet sich an der Adresse $0314 und zeigt normalerweise auf die Kernal-Adresse $FA65. Will man sich nun selbst in die IRQ-Routine einklinken, beispielsweise um eine eigene Sprite-Steuerung zu realisieren, oder weil man jede Sekunde die Rahmenfarbe wechseln will, so kann man dies auf diese Weise tun.

Dazu sollte man den IRQ-Vektor auf seine eigene Routine verbiegen und nach der Ausführung dieser Routine in die "restliche" Kernal-IRQ-Routine springen. Seien Sie aber vorsichtig, wenn Sie den IRQ-Vektor verbiegen. Die Interrupts müssen unbedingt verboten werden, da sonst bei einem evtl. auftretenden Interrupt der Rechner in die Wüste gehen kann...

Hier ist ein kleines Beispielprogramm, daß bei jedem 60. IRQ-Aufruf die Rahmenfarbe des 40-Zeichen-Bildschirmes um eins erhöht.

```
        ;Einschalten der IRQ-Routine
02000   78         SEI        ;Interrupts verbieten
02001   A9 0C      LDA #$0C   ;Lo-Byte der neuen IRQ-Routine
02003   8D 18 03   STA $0314  ;in Vektor speichern
02006   A9 20      LDA #$20   ;Hi-Byte der neuen IRQ-Routine
02008  $8D 19 03   STA $0315  ;in Vektor speichern
```

```
0200B    58        CLI        ;Interrupts wieder zulassen
0200C    E6 FD     INC $FD    ;Zähler erhöhen
0200E    A5 FD     LDA $FD    ;Hole Zähler
02010    C9 60     CMP #$3C   ;Bereits 60?
02012    D0 07     BNE $201B  ;Noch nicht erreicht
02014    EE 20 D0  INC $D020  ;Rahmenfarbe erhöhen
02017    A9 00     LDA #$00   ;Und Zähler wieder
02019    85 FD     STA $FD    ;auf Null setzen
0201B    4C 65 FA  JMP $FA65  ;Restliche IRQ-Routine
```

Eingeschaltet wird diese IRQ-Routine, indem man die Einschaltroutine an Adresse $2000 aufruft. Dies geschieht durch:

SYS DEC("2000")

Ab diesem Zeitpunkt wird der Rahmen in regelmäßigen Abständen die Farbe wechseln. Dies ist ein (wenn auch kleines) Beispiel, wie man sie die IRQ-Routine zu Nutze machen kann.

11.4 Ausschalten des BASIC-Interrupts

Wie bereits im Kapitel über den VIC-Chip erwähnt, kann es sehr ärgerlich sein, wenn der Interpreter einem immer "dazwischenfunkt". Aber auch hier gibt es Gegenmaßnahmen. Die Interrupts hören auf sich zu bewegen, wenn man dem Interpreter mitteilt, daß er die BASIC-IRQ-Routine nicht anzuspringen hat. Dies kann man an der Adresse $0A04 tun. Ist das Bit 0 gesetzt, so werden die BASIC-IRQ-Routinen für Grafik und Ton angesprungen. Löscht man dieses Bit, so wird augenblicklich damit aufgehört, die Sprites zu bewegen etc.

Dies ist eine willkommene Möglichkeite für alle Maschinenprogrammierer, die ihre Sprites selbst programmieren wollen. Allerdings ist der Text/Grafik-Modus hiervon nicht betroffen, dieser wird weiterhin automatisch umgeschaltet. Das liegt daran, daß dieses Umschalten in der Kernal-IRQ-Routine geschieht. Will man beispielsweise den Grafikmodus einschalten, dazu aber nicht die BASIC-Kommandos benutzen, so muß man entweder die entsprechenden Änderungen in den Zero-Page-Adressen

vornehmen, oder man muß in die Kernal-IRQ-Routine pfuschen.

Um nun den Effekt des Ausschaltens zu demonstrieren, definieren Sie ein Sprite und schalten Sie es ein:

```
SPRITE 1,1,2,0,1,1: REM Sprite 1 einschalten
MOVSPR 1,90#9     : REM Bewege Sprite 1
```

Wie auch immer Ihr Sprite aussehen mag - es bewegt sich jetzt auf jeden Fall über den Bildschirm. Versucht man nun in die VIC-Register zu schreiben, und Aussehen oder Position dieses Sprites zu ändern, so erkennt man lediglich ein kurzes Aufblitzen, und schon macht das Sprite wieder das, was es will, bzw. was das Betriebssystem will.

Durch Löschen des Bit 0 an Adresse $0A04 kann man diesen nervösen Bewegungen des Sprites allerdings ein für allemal Einhalt gebieten. Dies geschieht durch folgende Anweisung:

```
POKE DEC("0A04"), PEEK(DEC("0A04")) AND 254
```

Und schon bleibt es auf der Stelle stehen und bewegt sich nicht. Ab sofort sind Manipulationen am VIC-Chip mit Erfolg gekrönt. Zweite Möglichkeit, den Interrupt auszuschalten, ist folgendes Kommando:

```
POKE 216, 255
```

11.5 Positionieren des Cursors

Gerade von BASIC aus will man häufig den Cursor an eine beliebige Stelle im Bildschirm/Fenster positionieren. Leider existiert kein Kommando, daß das Positionieren des Cursors an eine beliebige Stelle ermöglicht. Lediglich den Grafik-Cursor können Sie mittels des Kommandos LOCATE an eine Position X,Y setzen. Durch Ausgabe von Cursor-Bewegungs-Codes ist die Positionierung zwar möglich, doch ist diese Methode

a) langsam,
b) speicherplatzaufwendig und
c) unelegant.

Wir bieten Ihnen nun eine Positionierungmöglichkeit an, indem die Kernal-Routine zum Setzen der Cursor-Position aufgerufen wird. Normalerweise wird im X-Register die Zeile des Cursors und im Y-Register die Spalte des Cursors im Fenster übergeben. Diese Parameter können Sie aber auch im SYS-Kommando als (optionale) Parameter übergeben.

Wie Sie dem Kernal-Listing entnehmen können, befindet sich die Routine zum Setzen der Cursor-Position ab Adresse $CC6A. Da wir die Cursor-Position setzen und nicht ermitteln wollen, können wir die Abfrage auf das Carry-Flag direkt am Anfang der Routine überspringen. Wir nehmen als Einsprungadresse $CC6C.

Die Syntax zum Positionieren des Cursores sieht wie folgt aus:

SYS DEC("CC6C"),,<Zeile>,<Spalte>

Die erste Zeile und die erste Spalte im Fenster ist jeweils die Zeile/Spalte Null.

Als Beispiel, wie Sie sich diese Positionierungroutine zunutze machen können, soll folgendes kleines Programm dienen:

```
10 REM **** DEMOPROGRAMM FÜR CURSOR-POSITIONIERUNG ****
20 :
30 SP=40 - 40*(PEEK(DEC("D7")): REM SPALTENZAHL 40 ODER 80?
40 PRINT CHR$(147);: REM BILDSCHIRM LÖSCHEN
50 X=INT(RND(TI)*24): REM ZEILE
60 Y=INT(RND(TI)*SP): REM SPALTE
70 SYS DEC("CC6C"),,X,Y
80 GET G$: IF G$="" THEN 50
90 END
```

11.6 Bootsektor und -routine

Wer von Ihnen schon einmal mit einem PC gearbeitet hat, weiß sicherlich den Vorteil eines BOOT-Sektors zu schätzen. Ein BOOT-Sektor (*BOOT* bedeutet im Englischen soviel wie "Stiefel".) Was hat aber ein Stiefel mit meinem hochmodernen Commodore 128 oder einem Rechner aus der PC-Serie zu tun, werden Sie sicherlich jetzt fragen. Die Antwort ist aber gar nicht so schwer. Der Stiefel als Fußbekleidung ist sozusagen der "unterste Teil" des Menschen. Er hat den eigenlichen Kontakt zu der Erde, auf der wir gehen und stehen. Nicht viel anders verhält es sich mit dem Bootsektor bei Ihrem Computer. Auch er ist der unterste Teil eines Programmes, die Verbindung zwischen dem Comuterprogramm und der Maschine selbst.

Wenn Sie Ihren Commodore einschalten, dann werden Sie sicherlich schon bemerkt haben - vorausgesetzt Sie haben ein Diskettenlaufwerk - daß dieses kurz nach dem Einschalten losläuft, ein paar maschinengewehrartige Geräusche von sich gibt, um dann anschließend zur Ruhe zu kommt. Auch wenn Sie eine Diskette eingelegt haben, immer läuft erst das Diskettenlaufwerk an, bevor sich der Rechner für seine eigentliche Arbeit zur Stelle meldet.

Die Ursache für dieses rechnergesteuerte Verhalten der Floppy ist der Versuch, den Bootsektor zu laden. Mit diesem Sektor läßt sich beispielsweise ein beliebiges Programm von der Diskette laden und starten, ohne daß Sie auch nur eine Taste an Ihrem Computer drücken müssen. Sie kennen dies sicherlich schon vom Booten des CP/M. Die Einsatzmöglichkeiten dieses "Stiefels" sind sehr vielfältig, um Sie aber voll nutzen zu können, ist es wichtig, den internen Aufbau des Sektors und den Ablauf der Bootroutine etwas näher zu beleuchten.

Da die vom Betriebssystem gesteuerte Bootroutine natürlich nicht die ganze Diskette nach einem solchen Sektor absuchen kann, gibt es nur eine fest definierte Stelle auf der Diskette, die als Bootsektor genutzt werden kann. Hierbei handelt es sich um:

Seite 0, Spur 01, Sektor 00

Doch Vorsicht: Auch wenn dieser Sektor physikalisch der allererste Datenblock auf einer Diskette ist, so kann es doch vorkommen, daß dieser Platz schon von anderen Dateien belegt ist. Sie sollten, bevor Sie einen Bootsektor auf einer Diskette installieren, immer erst nachprüfen, ob dieser bestimmte Sektor nicht schon belegt ist.

Um den Aufbau des Bootsektors verstehen zu können, sollten Sie den schematischen Ablauf der Bootroutine vor Augen haben. Diese im Kernal verankerte Routine läuft in den folgenden Schritten ab:

1.) Im DOS-Puffer der erweiterten Zeropage wird ein Block-Read Befehl auf die Spur 1, Sektor 0, aufgebaut.

2.) Der Befehl wird ausgeführt und der gelesene Block, sofern sich eine formatierte Diskette im Floppylaufwerk befindet, in den Kassettenpuffer des Rechners geladen.

3.) Die ersten drei Byte des gelesenen Blockes werden geprüft, ob sie den erforderlichen Identifizierungscode für einen Bootsektor enthalten. Dieser Identifizierungscode lautet: *CBM*. Ist dieser Code nicht vorhanden, so wird die Bootroutine abgebrochen.

4.) Die auf den CBM-Code folgenden vier Byte werden in vier Zeropagezeiger geladen. Normalerweise sind diese vier Byte auf den Wert $00 gesetzt. Die ersten beiden Byte können eine Startadresse enthalten, die jedoch nichts mit der Adresse zu tun hat, an die etwa das geswünschte Programm geladen werden soll. Das 3. Byte ist der entsprechende Konfigurationsindex zu der genannten Startadresse. Doch alle Einträge in den ersten drei Byte sind ohne Interesse, wenn das vierte Byte den Wert $00 aufweist. Dieses Byte enthält die Zahl an Blöcken, die außer dem eigentlichen Bootsektor noch von der Diskette nachgeladen werden sollen.

5.) Unabhängig davon, ob der Blockzähler im BOOT-Block gesetzt ist oder nicht, werden erst einmal die auf diese vier Adreß- und Steuerbytes folgenden Bytes gelesen und über die BSOUT Routine auf dem Bildschirm ausgegeben. Hiermit läßt sich der Bildschirm löschen oder eine beliebige Einschaltmeldung auf dem Bildschirm ausgeben. Diese Zeichenausgabe auf den Bildschirm erfolgt so lange, bis der Code $00 gefunden wird.

6.) Jetzt bekommen die unter Punkt 4 gelesenen Steuerbytes eine Bedeutung. Ist der Blockzähler auf Null gesetzt, so wird diese Routine übersprungen. Ist das aber nicht der Fall, dann wird im DOS-Befehlspuffer ein neuer Befehlsstring gebildet, der dann einen weiteren BOOT-Block von der Diskette nachladen soll. Die Bestimmung des Folgebootblocks fällt dabei denkbar einfach aus: Die Sektornummer wird um den Faktor 1 erhöht. Ist die Sektornummer größer 20 (es gibt nur maximal 21 Sektoren pro Spur mit den Nummern 0 -20), so wird die Spurnummer um 1 erhöht und die Sektornummer wieder auf $00 heruntergesetzt. Dieser gebildete Block-Read Befehl wird nun ausgeführt, wobei der gelesene Block an die durch die ersten drei Byte gebidete Adresse und Konfiguration gespeichert wird. Die Speicheradresse für die Folgebootblöcke wird erhöht und der Blockzähler wird um 1 vermindert. Das geschieht so lange, bis dieser Blockzähler auf Null heruntergezählt ist.

7.) Anschließend setzt die Bootroutine wieder hinter der (wenn vorhanden) Textkonstanten des ursprünglichen BOOT-Blocks im Kassettenpuffer auf. Hier kann nun ein Dateiname, wie er auch im Directory der Diskette verzeichnet ist, stehen. Mit Ausnahme der Tatsache, daß die den Dateinamen bildenden Zeichen nicht auf dem Bildschirm ausgeben werden, werden auch hier alle Bytes gelesen, bis die Bootroutine abermals auf das Endezeichen $00 stößt. Die Länge des Dateinamens wird dabei in einem Zähler festgehalten.

8.) Jetzt kommt wieder eine Entscheidungsmöglichkeit. Ist die Länge des Dateinamens im Zähler mit einem Wert ungleich Null festgehalten, so werden dem Dateinamen die beiden Zeichen: "0:" vorangestellt, der Dateinamenzähler um diese beiden Zeichen erhöht und es wird in die Kernal LOAD-Routine verzweigt, um das so gekennzeichnete Programm in den Speicher des Rechners zu laden. Wenn das geschehen ist, oder wenn die Länge des Dateinamnes mit Null festgehalten wurde, so setzt die Bootroutine wieder hinter dem $00 Trenncode auf, der das Ende des Dateinamens kennzeichnet.

9.) Die auf den Dateinamen folgenden Bytes werden nun als ein Maschinenprogramm des Bootsektors angesehen und die Bootroutine übergibt die Steuerung an diese Maschinenprogramm. Von dieser Stelle an sind Sie selbst verantwortlich dafür, ob das nachgeladene Programm gestartet werden soll oder ob ein weiteres Programm über eine eigene Routine nachgeladen werden soll oder ob in einen der eventuell nachgeladenen Folgebootblöcke verzweigt werden soll.

Wenn Sie bei der Erstellung eigener Bootsektoren die oben genannten Schritte des Betriebssystems beachten, so werden Sie sehr schnell feststellen, daß es gar nicht so schwer ist, eigene Bootsektoren zu erzeugen, wenn man nur weiß, was man dabei beachten muß. Hier noch einaml die wichtigsten Merkmale und Anordnungen:

Byte 0, 1, 2:	CBM Identifikationscode.
Byte 3, 4:	Speicheradresse für Folgebootsektoren.
Byte 5:	Konfigurationsindex für Folgebootsektoren.
Byte 6:	Blockzähler für die Anzahl der Folgebootsektoren.
Byte 7:	Bis zum 1.Trenncode ($00) eigene Bootnachricht.
Dann:	Name des nachzuladenden Pogramms gefolgt von $00.
Dann:	Eigener Maschinenenprogrammeinsprung.

Adresse des Bootsektors: Seite 0, Spur 1, Sektor 0

Anhang

Die Zeichensätze

Durch Betätigen der ASCII/DIN-Taste können Sie zwischen den beiden Zeichensätzen hin- und herschalten. Sie sehen die Auswirkungen sofort auf dem Monitor. Diese Taste wird per Interrupt abgefragt, d.h. daß bei Betätigen der Taste sofort in die entsprechende Maschinenroutine verzweigt wird. Sie merken dies daran, daß auf dem 40-Zeichen-Bildschirm der Zeichenwechsel sofort stattfindet und der Rechner für ca. 1 Sekunde gesperrt ist. Das liegt daran, daß der Rechner den neu definierten Zeichensatz in den Speicher des VDC (80-Zeichen-Bildschirm, siehe auch entsprechendes Kapitel) erst einmal kopieren muß, da dieser seine Zeichen *nicht* aus den ROM bezieht.

Physikalisch liegen die beiden Zeichensätze ASCII und DIN an derselben Adresse, nämlich an $D000. Wenn die ASCII/DIN-Taste betätigt wird, so werden die beiden Zeichensätze für den VIC hardwaremäßig umgeschaltet.

Auf den nun folgenden Seiten finden Sie die beiden Zeichensätze vor. Sie erhalten Informationen über die Adresse, ab der die Matrix der ausgedruckten Zeichens abgelegt ist, sowie in Klammern den POKE-Code.

Um nicht unnötigen Platz zu vergeuden, haben wir auf den Ausdruck der ebenfalls definierten reversen Zeichen verzichtet. Die Adresse dieser Zeichen erhalten Sie, indem Sie auf die angegebene Adresse des entsprechenden Zeichens $0400 addieren.

Den Zeichensatz für den 80-Zeichen-Controller können Sie sehr leicht ändern, indem Sie die entsprechenden Adressen im VDC-RAM direkt manipulieren. Hierzu weitere Informationen im Kapitel 5 (Für den VIC-Chip siehe Kapitel 2).

D000 (000)	D008 (001)	D010 (002)	D018 (003)
3C	18	7C	3C
66	3C	66	66
6E	66	66	60
6E	7E	7C	60
60	66	66	60
62	66	66	66
3C	66	7C	3C
00	00	00	00

D020 (004)	D028 (005)	D030 (006)	D038 (007)
78	7E	7E	3C
6C	60	60	66
66	60	60	60
66	78	78	6E
66	60	60	66
6C	60	60	66
78	7E	60	3C
00	00	00	00

D040 (008)	D048 (009)	D050 (010)	D058 (011)
66	3C	1E	66
66	18	0C	6C
66	18	0C	78
7E	18	0C	70
66	18	0C	78
66	18	6C	6C
66	3C	38	66
00	00	00	00

D060 (012)	D068 (013)	D070 (014)	D078 (015)
60	63	66	3C
60	77	76	66
60	7F	7E	66
60	6B	7E	66
60	63	6E	66
60	63	66	66
7E	63	66	3C
00	00	00	00

D080 (016)	D088 (017)	D090 (018)	D098 (019)
7C	3C	7C	3C
66	66	66	66
66	66	66	60
7C	66	7C	3C
60	66	78	06
60	3C	6C	66
60	0E	66	3C
00	00	00	00

D0A0 (020)	D0A8 (021)	D0B0 (022)	D0B8 (023)
7E	66	66	63
18	66	66	63
18	66	66	63
18	66	66	6B
18	66	66	7F
18	66	3C	77
18	3C	18	63
00	00	00	00

D0C0 (024)	D0C8 (025)	D0D0 (026)	D0D8 (027)
66	66	7E	3C
66	66	06	30
3C	66	0C	30
18	3C	18	30
3C	18	30	30
66	18	60	30
66	18	7E	3C
00	00	00	00

D0E0 (028)	D0E8 (029)	D0F0 (030)	D0F8 (031)
0C	3C	00	00
12	0C	18	10
30	0C	3C	30
7C	0C	7E	7F
30	0C	18	7F
62	0C	18	30
FC	3C	18	10
00	00	18	00

D100 (032)	D108 (033)	D110 (034)	D118 (035)
00	18	66	66
00	18	66	66
00	18	66	FF
00	18	00	66
00	00	00	FF
00	00	00	66
00	18	00	66
00	00	00	00

D120 (036)	D128 (037)	D130 (038)	D138 (039)
18	62	3C	06
3E	66	66	0C
60	0C	3C	18
3C	18	38	00
06	30	67	00
7C	66	66	00
18	46	3F	00
00	00	00	00

D140 (040)	D148 (041)	D150 (042)	D158 (043)
0C	30	00	00
18	18	66	18
30	0C	3C	18
30	0C	FF	7E
30	0C	3C	18
18	18	66	18
0C	30	00	00
00	00	00	00

D160 (044)	D168 (045)	D170 (046)	D178 (047)
00	00	00	00
00	00	00	03
00	00	00	06
00	7E	00	0C
00	00	00	18
18	00	18	30
18	00	18	60
30	00	00	00

D180 (048)	D188 (049)	D190 (050)	D198 (051)
3C	18	3C	3C
66	18	66	66
6E	38	06	06
76	18	0C	1C
66	18	30	06
66	18	60	66
3C	7E	7E	3C
00	00	00	00

D1A0 (052)	D1A8 (053)	D1B0 (054)	D1B8 (055)
06	7E	3C	7E
0E	60	66	66
1E	7C	60	0C
66	06	7C	18
7F	06	66	18
06	66	66	18
06	3C	3C	18
00	00	00	00

D1C0 (056)	D1C8 (057)	D1D0 (058)	D1D8 (059)
3C	3C	00	00
66	66	00	00
66	66	18	18
3C	3E	00	00
66	06	00	00
66	66	18	18
3C	3C	00	18
00	00	00	30

D1E0 (060)	D1E8 (061)	D1F0 (062)	D1F8 (063)
0E	00	70	3C
18	00	18	66
30	7E	0C	06
60	00	06	0C
30	7E	0C	18
18	00	18	00
0E	00	70	18
00	00	00	00

D200 (064)	D208 (065)	D210 (066)	D218 (067)
00	08	18	00
00	1C	18	00
00	3E	18	00
FF	7F	18	FF
FF	7F	18	FF
00	1C	18	00
00	3E	18	00
00	00	18	00

D220 (068)	D228 (069)	D230 (070)	D238 (071)
00	00	00	30
00	FF	00	30
FF	FF	00	30
FF	00	00	30
00	00	FF	30
00	00	FF	30
00	00	00	30
00	00	00	30

D240 (072)	D248 (073)	D250 (074)	D258 (075)
0C	00	18	18
0C	00	18	18
0C	00	1C	38
0C	E0	0F	F0
0C	F0	07	E0
0C	38	00	00
0C	18	00	00
0C	18	00	00

D260 (076)	D268 (077)	D270 (078)	D278 (079)
C0	C0	03	FF
C0	E0	07	FF
C0	70	0E	C0
C0	38	1C	C0
C0	1C	38	C0
C0	0E	70	C0
FF	07	E0	C0
FF	03	C0	C0

D280 (080)	D288 (081)	D290 (082)	D298 (083)
FF	00	00	36
FF	3C	00	7F
03	7E	00	7F
03	7E	00	7F
03	7E	00	3E
03	7E	FF	1C
03	3C	FF	08
03	00	00	00

D2A0 (084)	D2A8 (085)	D2B0 (086)	D2B8 (087)
60	00	C3	00
60	00	E7	3C
60	00	7E	7E
60	07	3C	66
60	0F	3C	66
60	1C	7E	7E
60	18	E7	3C
60	18	C3	00

D2C0 (088)	D2C8 (089)	D2D0 (090)	D2D8 (091)
18	06	08	18
18	06	1C	18
66	06	3E	18
66	06	7F	FF
18	06	3E	FF
18	06	1C	18
3C	06	08	18
00	06	00	18

D2E0 (092)	D2E8 (093)	D2F0 (094)	D2F8 (095)
C0	18	00	FF
C0	18	00	7F
30	18	03	3F
30	18	3E	1F
C0	18	76	0F
C0	18	36	07
30	18	36	03
30	18	00	01

```
D300      (096)   D308      (097)   D310      (098)   D318      (099)
00                F0                00                FF
00                F0                00                00
00                F0                00                00
00                F0                00                00
00                F0                FF                00
00                F0                FF                00
00                F0                FF                00
00                F0                FF                00

D320      (100)   D328      (101)   D330      (102)   D338      (103)
00                C0                CC                03
00                C0                CC                03
00                C0                33                03
00                C0                33                03
00                C0                CC                03
00                C0                CC                03
00                C0                33                03
FF                C0                33                03

D340      (104)   D348      (105)   D350      (106)   D358      (107)
00                FF                03                18
00                FE                03                18
00                FC                03                18
00                F8                03                1F
CC                F0                03                1F
CC                E0                03                18
33                C0                03                18
33                80                03                18

D360      (108)   D368      (109)   D370      (110)   D378      (111)
00                18                00                00
00                18                00                00
00                18                00                00
00                1F                F8                00
0F                1F                F8                00
0F                00                18                00
0F                00                18                FF
0F                00                18                FF

D380      (112)   D388      (113)   D390      (114)   D398      (115)
00                18                00                18
00                18                00                18
00                18                00                18
1F                FF                FF                F8
1F                FF                FF                F8
18                00                18                18
18                00                18                18
18                00                18                18

D3A0      (116)   D3A8      (117)   D3B0      (118)   D3B8      (119)
C0                E0                07                FF
C0                E0                07                FF
C0                E0                07                00
C0                E0                07                00
C0                E0                07                00
C0                E0                07                00
C0                E0                07                00
C0                E0                07                00
```

D3C0 (120)	D3C8 (121)	D3D0 (122)	D3D8 (123)
FF	00	03	00
FF	00	03	00
FF	00	03	00
00	00	03	00
00	00	03	F0
00	FF	03	F0
00	FF	FF	F0
00	FF	FF	F0

D3E0 (124)	D3E8 (125)	D3F0 (126)	D3F8 (127)
0F	18	F0	F0
0F	18	F0	F0
0F	18	F0	F0
0F	F8	F0	F0
00	F8	00	0F
00	00	00	0F
00	00	00	0F
00	00	00	0F

D000	(000)	3C	42	38	24	1C	42	3C	00
D008	(001)	18	24	42	7E	42	42	42	00
D010	(002)	7C	22	22	3C	22	22	7C	00
D018	(003)	1C	22	40	40	40	22	1C	00
D020	(004)	78	24	22	22	22	24	78	00
D028	(005)	7E	40	40	78	40	40	7E	00
D030	(006)	7E	40	40	78	40	40	40	00
D038	(007)	1C	22	40	4E	42	22	1C	00
D040	(008)	42	42	42	7E	42	42	42	00
D048	(009)	1C	08	08	08	08	08	1C	00
D050	(010)	0E	04	04	04	04	44	38	00
D058	(011)	42	44	48	70	48	44	42	00
D060	(012)	40	40	40	40	40	40	7E	00
D068	(013)	42	66	5A	5A	42	42	42	00
D070	(014)	42	62	52	4A	46	42	42	00
D078	(015)	18	24	42	42	42	24	18	00
D080	(016)	7C	42	42	7C	40	40	40	00
D088	(017)	18	24	42	42	4A	24	1A	00
D090	(018)	7C	42	42	7C	48	44	42	00
D098	(019)	3C	42	40	3C	02	42	3C	00
D0A0	(020)	3E	08	08	08	08	08	08	00
D0A8	(021)	42	42	42	42	42	42	3C	00
D0B0	(022)	42	42	42	24	24	18	18	00
D0B8	(023)	42	42	42	5A	5A	66	42	00

D0C0 (024)	D0C8 (025)	D0D0 (026)	D0D8 (027)
42	22	7E	3C
42	22	02	20
24	22	04	20
18	1C	18	20
24	08	20	20
42	08	40	20
42	08	7E	3C
00	00	00	00

D0E0 (028)	D0E8 (029)	D0F0 (030)	D0F8 (031)
00	3C	10	00
40	04	38	00
20	04	54	00
10	04	10	00
08	04	10	00
04	04	10	00
02	3C	10	00
00	00	00	FF

D100 (032)	D108 (033)	D110 (034)	D118 (035)
00	08	24	24
00	08	24	24
00	08	24	7E
00	08	00	24
00	00	00	7E
00	00	00	24
00	08	00	24
00	00	00	00

D120 (036)	D128 (037)	D130 (038)	D138 (039)
08	00	30	08
1E	62	48	08
28	64	48	08
1C	08	30	00
0A	10	4A	00
3C	26	44	00
08	46	3A	00
00	00	00	00

D140 (040)	D148 (041)	D150 (042)	D158 (043)
04	20	08	00
08	10	2A	08
10	08	1C	08
10	08	3E	3E
10	08	1C	08
08	10	2A	08
04	20	08	00
00	00	00	00

D160 (044)	D168 (045)	D170 (046)	D178 (047)
00	00	00	00
00	00	00	02
00	00	00	04
00	7E	00	08
00	00	00	10
08	00	18	20
08	00	18	40
10	00	00	00

Adresse	Nr.	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
D180	(048)	3C	42	46	5A	62	42	3C	00
D188	(049)	08	18	28	08	08	08	3E	00
D190	(050)	3C	42	02	0C	30	40	7E	00
D198	(051)	3C	42	02	1C	02	42	3C	00
D1A0	(052)	04	0C	14	24	7E	04	04	00
D1A8	(053)	7E	40	78	04	02	44	38	00
D1B0	(054)	1C	20	40	7C	42	42	3C	00
D1B8	(055)	7E	42	04	08	10	10	10	00
D1C0	(056)	3C	42	42	3C	42	42	3C	00
D1C8	(057)	3C	42	42	3E	02	04	38	00
D1D0	(058)	00	00	08	00	00	08	00	00
D1D8	(059)	00	00	08	00	00	08	08	10
D1E0	(060)	0E	18	30	60	30	18	0E	00
D1E8	(061)	00	00	7E	00	7E	00	00	00
D1F0	(062)	70	18	0C	06	0C	18	70	00
D1F8	(063)	3C	42	02	0C	10	00	10	00
D200	(064)	10	08	04	00	00	00	00	00
D208	(065)	FF	7F	3F	1F	0F	07	03	01
D210	(066)	FF	FF	00	00	00	00	00	00
D218	(067)	00	00	00	00	FF	FF	FF	FF
D220	(068)	00	00	00	00	0F	0F	0F	0F
D228	(069)	0F	0F	0F	0F	00	00	00	00
D230	(070)	08	08	08	04	03	00	00	00
D238	(071)	08	08	08	10	E0	00	00	00

D240 (072)	D248 (073)	D250 (074)	D258 (075)
07	C0	03	80
07	C0	03	40
07	C0	03	20
07	C0	03	10
07	C0	03	08
07	C0	03	04
07	C0	03	02
07	C0	03	01

D260 (076)	D268 (077)	D270 (078)	D278 (079)
80	01	FF	FF
80	02	00	80
80	04	00	80
80	08	00	80
80	10	00	80
80	20	00	80
80	40	00	80
FF	80	00	80

D280 (080)	D288 (081)	D290 (082)	D298 (083)
FF	FF	00	00
01	FE	00	00
01	FC	00	00
01	F8	00	00
01	F0	03	F0
01	E0	04	F0
01	C0	08	F0
01	80	08	F0

D2A0 (084)	D2A8 (085)	D2B0 (086)	D2B8 (087)
00	E0	FF	F0
00	E0	FF	F0
00	E0	FF	F0
00	E0	00	F0
E0	E0	00	00
10	E0	00	00
08	E0	00	00
08	E0	00	00

D2C0 (088)	D2C8 (089)	D2D0 (090)	D2D8 (091)
00	00	F0	00
00	00	F0	00
00	00	F0	00
00	00	F0	00
00	00	F0	AA
FF	00	F0	55
FF	FF	F0	AA
FF	FF	F0	55

D2E0 (092)	D2E8 (093)	D2F0 (094)	D2F8 (095)
01	AA	00	00
01	55	00	00
01	AA	01	00
01	55	3E	00
01	AA	54	00
01	55	14	00
01	AA	14	00
FF	55	00	FF

D300 (096)	D308 (097)	D310 (098)	D318 (099)
00	08	08	08
00	08	08	08
00	08	08	08
00	08	08	08
00	08	0F	FF
00	08	00	00
00	08	00	00
00	08	00	00

D320 (100)	D328 (101)	D330 (102)	D338 (103)
08	08	00	08
08	08	00	08
08	08	00	08
08	08	00	08
F8	0F	FF	F8
00	08	00	08
00	08	00	08
00	08	00	08

D340 (104)	D348 (105)	D350 (106)	D358 (107)
00	00	00	08
00	00	00	08
00	00	00	08
00	00	00	08
0F	FF	F8	FF
08	08	08	08
08	08	08	08
08	08	08	08

D360 (108)	D368 (109)	D370 (110)	D378 (111)
08	18	10	08
10	24	08	10
3C	20	3C	20
42	70	42	00
7E	20	7E	00
40	21	40	00
3C	5E	3C	00
00	00	00	00

D380 (112)	D388 (113)	D390 (114)	D398 (115)
1C	00	20	10
22	00	10	08
4A	42	38	42
56	42	04	42
4C	42	3C	42
20	46	44	46
1E	BA	3A	3A
00	80	00	00

D3A0 (116)	D3A8 (117)	D3B0 (118)	D3B8 (119)
10	08	08	08
28	14	14	14
38	3C	08	3C
04	42	08	42
3C	7E	08	42
44	40	08	42
3A	3C	1C	3C
00	00	00	00

```
D3C0          (120)
08 □□□□■□□□
14 □□□■□■□□
42 □■□□□□■□
42 □■□□□□■□
42 □■□□□□■□
46 □■□□□■■□
3A □□■■■□■□
00 □□□□□□□□

D3C8          (121)
1F □□□■■■■■
10 □□□■□□□□
10 □□□■□□□□
10 □□□■□□□□
D0 ■■□■□□□□
30 □□■■□□□□
10 □□□■□□□□
00 □□□□□□□□

D3D0          (122)
7F □■■■■■■■
21 □□■□□□□■
10 □□□■□□□□
08 □□□□■□□□
10 □□□■□□□□
21 □□■□□□□■
7F □■■■■■■■
00 □□□□□□□□

D3D8          (123)
5A □■□■■□■□
24 □□■□□■□□
42 □■□□□□■□
7E □■■■■■■□
42 □■□□□□■□
42 □■□□□□■□
42 □■□□□□■□
00 □□□□□□□□

D3E0          (124)
5A □■□■■□■□
24 □□■□□■□□
42 □■□□□□■□
42 □■□□□□■□
42 □■□□□□■□
24 □□■□□■□□
18 □□□■■□□□
00 □□□□□□□□

D3E8          (125)
18 □□□■■□□□
42 □■□□□□■□
42 □■□□□□■□
42 □■□□□□■□
42 □■□□□□■□
42 □■□□□□■□
3C □□■■■■□□
00 □□□□□□□□

D3F0          (126)
3C □□■■■■□□
42 □■□□□□■□
42 □■□□□□■□
5C □■□■■■□□
42 □■□□□□■□
42 □■□□□□■□
5C □■□■■■□□
40 □■□□□□□□

D3F8          (127)
08 □□□□■□□□
14 □□□■□■□□
00 □□□□□□□□
00 □□□□□□□□
00 □□□□□□□□
00 □□□□□□□□
00 □□□□□□□□
00 □□□□□□□□
```

Die Tastaturmatrix

Die Tastatur Ihres Rechner ist in Form einer Matrix aufgebaut. Diese müssen Sie sich als ein Netz von Leitungen vorstellen: in der waagerechten Ebene haben Sie 11 Leitungen und in der senkrechten 8 Leitungen. Wenn Sie nun eine Taste betätigen, schließen Sie somit den normalerweise offenen Kontakt zwischen einer waagerechten und einer senkrechten Leitung. Daraus kann der Rechner erkennen, welche Taste gedrückt wurde.

Soweit das Prinzip der Tastaturmatrix. In Wirklichkeit geht es doch eine ganze Ecke komplizierter zu, da nicht für jede der 11 waagerechten und 8 senkrechten Leitungen ein eigener Anschluß an einem Ein-/Ausgabebaustein zur Verfügung steht. Im Commodore 128 gibt es 2 Bausteine mit insgesamt drei Ports, die diese Aufgabe der Tastatur-Matrixabfrage bewältigen müssen. Am Cia 1 stehen dafür die Leitungen PA0 - PA7 und die Leitungen PB0 - PB7 zur Verfügung. Diese 16 Leitungen können wahlweise auf Ein- oder Ausgabe programmiert werden. Theoretisch ist es also möglich, auch 16 Bit Wert über diese Leitung zu übertragen. Die Leitungen PA0 bis PA7 sind verantwortlich für die ersten 8 Matrixzeilen der Tastaturverdrahtung. Die fehlenden 3 Leitungen sind, Sie werden es nicht glauben, am VIC-Chip angeschlossen.

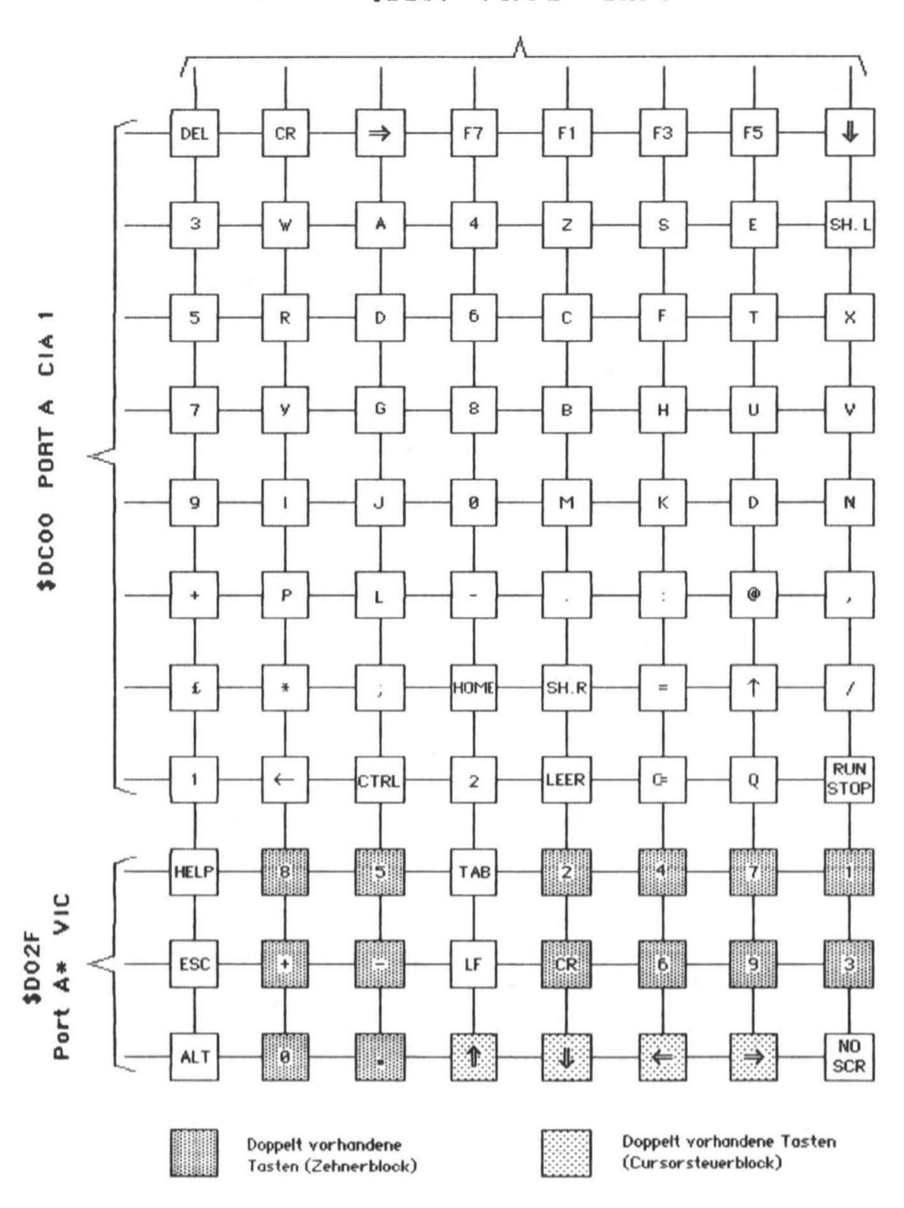
Der in Ihrem Commodore eingebaute VIC-Chip hat gegenüber dem im Commodore 64 benutzten Baustein 2 zusätzliche Register. Das ist einerseits das Register mit der Adresse $D030, welches beispielsweise für die Taktfrequenz, mit der der Rechner arbeiten soll (1 oder 2 Megaherz), verantwortlich ist. Dieses Register soll uns aber hier nicht interessieren. Das andere, neu hinzugekommene Register, hat die Adresse $D02F. Über dieses Register werden die zusätzlichen drei Tastaturmatrixzeilen abgefragt. Das Register stellt uns dafür die Bits 0 - 3 zur Verfügung, benutzt werden aber nur die Bits 0 - 2, da nur drei zusätzliche Matrixzeilen abgefragt werden müssen. Die 8 Matrixspalten werden über die Leitungen PB0 - PB7 des CIA 1 über den Port B angesprochen.

Die eigentliche Tastaturabfrage verläuft nach folgendem Muster: Der Port A des CIA 1 (Leitungen PA0 - PA7) werden auf Lo-Value Wert gelegt, d.h., das Register wird mit dem Hexwert 00

geladen. Des weiteren wird, um auch die restlichen drei Matrixzeilen abzuprüfen, das zuständige VIC-Register ebenfalls mit einem Lo-Value Wert vorgeladen. Anschließend wird der auf Eingang geschaltete Port B des CIA 1 (Leitungen PB0 - PB7) abgefragt. Wenn an irgendeiner Stelle eine Taste gedrückt ist, so wir damit auch eine der Eingangsleitungen am Port B auf den Lo-Pegel geschaltet, welches sich dadurch bemerkbar macht, daß der am Port B ausgelesene Wert ungleich High-Value ($FF) ist. Zu diesem Zeitpunkt kann nur erkannt werden, ob eine Taste gedrückt wurde. Welche das ist, das kann noch nicht bestimmt werden.

Die genaue Position innerhalb der Tastaturmatrix wird dann, wenn ein Tastendruck registriert wurde, in der Art ermittelt, daß jede der 11 Matrixzeilen einmal kurz auf den Lo-Value Pegel gelegt wird und anschließend sofort der immer noch auf Eingang geschaltete Port B ausgelesen wird. Nun kann erkannt werden, in welcher Zeile und in welcher Spalte der Matrix eine Taste gedrückt wurde. Bei jeder Überprüfung, sowohl der Zeilen, als auch der Spalten läuft ein Zählregister mit, in welchem hinterher die physikalische Anordnungsnummer der gedrückten Taste mitgezählt wird. Acuh die Abfrage der Joy-stickport erfolgt auf die gleiche Weise, wie die normale Tastaturabfrage, da die Anschlüsse der Joystickports mit einigen Tasten des Keyboards *parallel* geschaltet sind.

Auf dem Schaubild der nächsten Seite können sie die physikalische Anordnung der Tasten und ihre Anschlüsse an die drei Ports erkennen. Intersessant ist hier sicherlich noch die Tatsache, daß die Tasten des Zehnerblocks zwar hinterher auf dem Bildschirm das gleiche Ergebnis liefern, dennoch aber eindeutig von den normalen Zifferntasten unter-schieden werden können. Das gilt ebenso für die Cursorsteu-ertasten und die weiteren Tasten, die an Ihrem Rechner scheinbar doppelt vorhanden sind.



$DC01 Port B CIA 1
$DC00 PORT A CIA 1
DEL
CR
⇒
F7
F1
F3
F5
⇓
3
W
A
4
Z
S
E
SH. L
5
R
D
6
C
F
T
X
7
Y
G
8
B
H
U
V
9
I
J
0
M
K
O
N
+
P
L
-
.
:
@
,
£
*
;
HOME
SH.R
=
↑
/
1
←
CTRL
2
LEER
C=
Q
RUN STOP
$D02F Port A* VIC
HELP
8
5
TAB
2
4
7
1
ESC
+
-
LF
CR
6
9
3
ALT
0
.
⇑
⇓
⇐
⇒
NO SCR
Doppelt vorhandene Tasten (Zehnerblock)
Doppelt vorhandene Tasten (Cursorsteuerblock)

Die verschiedenen Rechner-Modi

Ihnen ist sicherlich auch schon bekannt, daß der Commodore 128 drei verschiedene Rechner-Modi hat, die sich nicht nur durch verschiedene Programmierungsarten auszeichnen, sondern auch dadurch, daß in einigen Modi verschiedene Bausteine nicht benötigt werden.

Sie können einen der drei folgenden Modi auswählen:

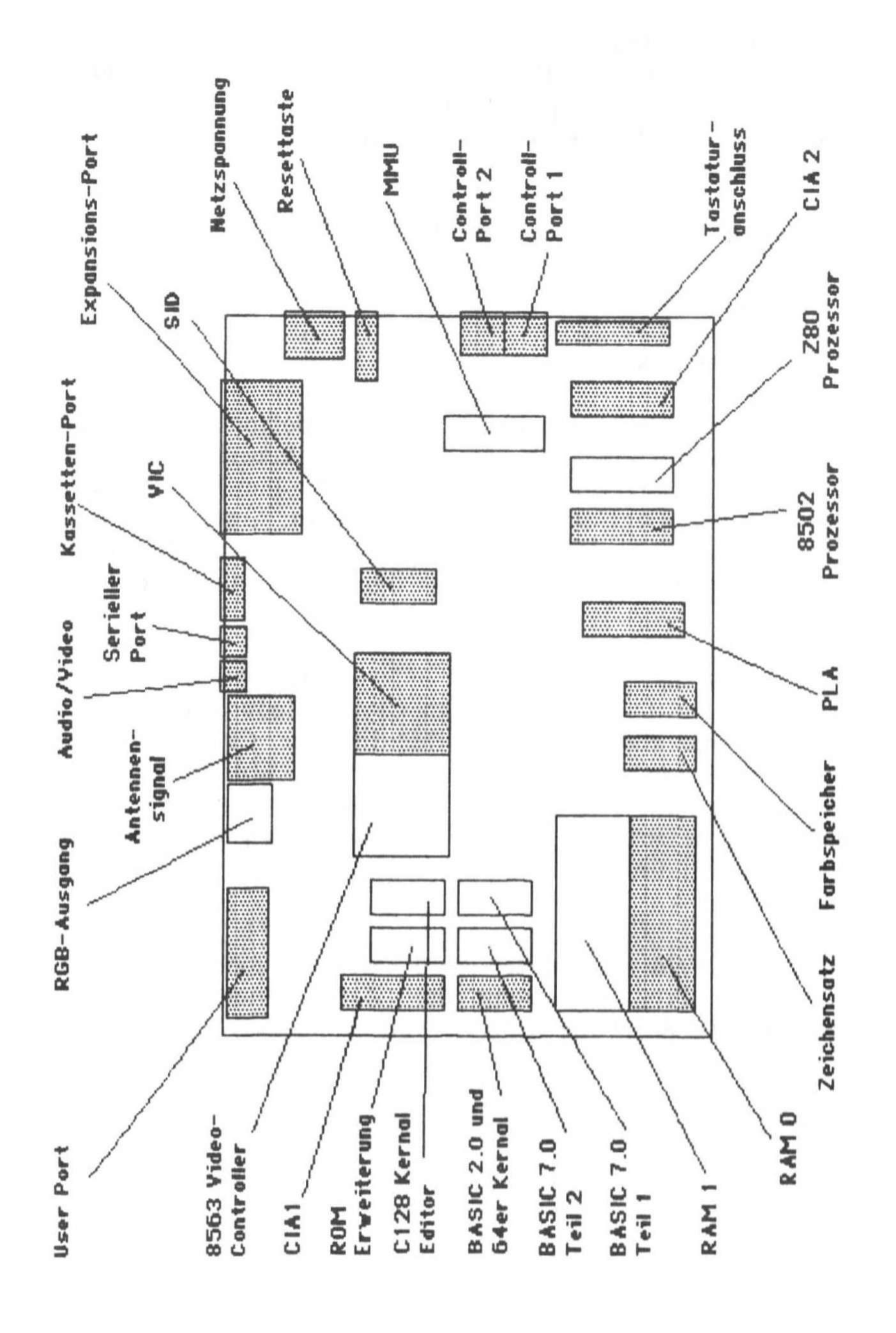
- *CP/M Plus*
- *Commodore 128*
- *Commodore 64*

Um Ihnen nun möglichst übersichtlich zu demonstrieren, welche Bausteine wann ein- und wann ausgeschaltet sind, haben wir drei Grafiken angefertigt. Dabei ist jeder Baustein Bezeichnet und Grau hinterlegt, sollte er in Aktion sein. Ausgeschaltet heißt natürlich lediglich, daß die MMU den Zugriff zu diesen Baustein sperrt. Die Z-80 ist auch nur *angehalten*, nicht ausgeschaltet.

Übrigens ist der CP/M-Modus interessanter als Sie vielleicht denken mögen. Die 1571 wird im CP/M-Modus noch eine Idee schneller und Sie haben die Möglichkeit CP/M-Programme laufen zu lassen oder CP/M zu erlernen, was auch nicht zu unterschätzen ist. Die 1571 ist auch in der Lage, verschiedene Fremdformate zu lesen, wie beispielsweise original OSBORNE-Disketten - dies macht einen Programmaustausch sehr interessant und einfach.

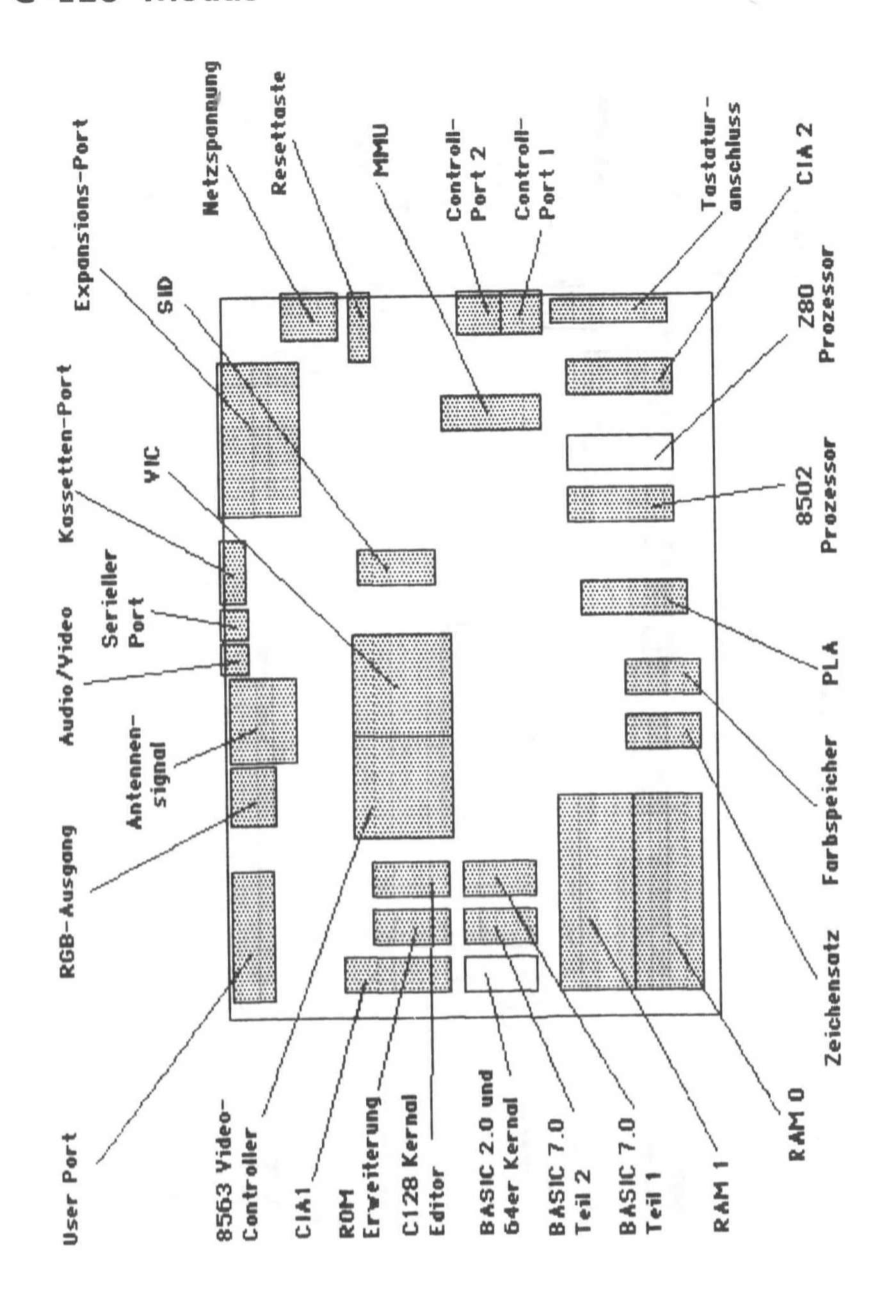
Bezüglich CP/M gibt es genügend Literatur - auch in unserem Haus. Die Programmierung der Z-80 unter dem CP/M-Assembler ist sicherlich recht reizvoll!

C-64-Modus

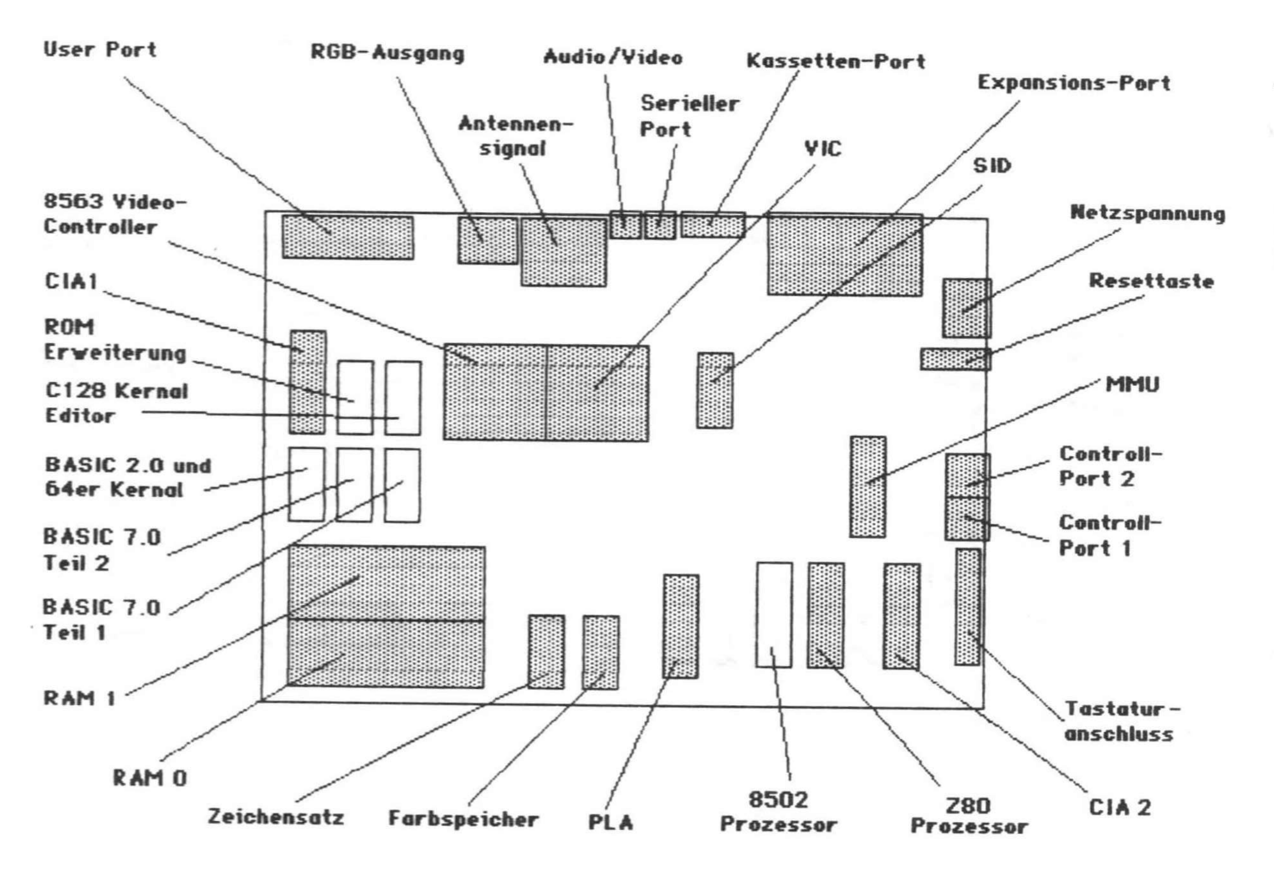


Expansions-Port
Netzspannung
Resettaste
MMU
Controll-Port 2
Controll-Port 1
Tastatur-anschluss
CIA 2
SID
Z80 Prozessor
Kassetten-Port
VIC
8502 Prozessor
Serieller Port
Audio/Video
PLA
Antennen-signal
RGB-Ausgang
Farbspeicher
Zeichensatz
User Port
8563 Video-Controller
CIA1
ROM Erweiterung
C128 Kernal Editor
BASIC 2.0 und 64er Kernal
BASIC 7.0 Teil 2
BASIC 7.0 Teil 1
RAM 1
RAM 0

C-128-Modus


Expansions-Port
Netzspannung
Resettaste
MMU
Controll-
Port 2
Controll-
Port 1
Tastatur-
anschluss
CIA 2
Z80
Prozessor
8502
Prozessor
PLA
Farbspeicher
Zeichensatz
SID
VIC
Kassetten-Port
Serieller
Port
Audio/Video
Antennen-
signal
RGB-Ausgang
User Port
8563 Video-
Controller
CIA1
ROM
Erweiterung
C128 Kernal
Editor
BASIC 2.0 und
64er Kernal
BASIC 7.0
Teil 2
BASIC 7.0
Teil 1
RAM 1
RAM 0

CP/M-Modus



User Port
RGB-Ausgang
Audio/Video
Kassetten-Port
Expansions-Port
Antennen-signal
Serieller Port
VIC
SID
8563 Video-Controller
Netzspannung
CIA1
Resettaste
ROM Erweiterung
C128 Kernal Editor
MMU
BASIC 2.0 und 64er Kernal
Controll-Port 2
Controll-Port 1
BASIC 7.0 Teil 2
BASIC 7.0 Teil 1
RAM 1
Tastatur-anschluss
RAM 0
Zeichensatz
Farbspeicher
PLA
8502 Prozessor
Z80 Prozessor
CIA 2

Die verschiedenen Einschaltmodi

Auf den vorangegangenen Seiten finden Sie die drei Rechnermodi grafisch aufbereitet dargestellt. Es gibt neben des *GO 64* aber noch eine andere Möglichkeit, den C64-Modus zu betreten. Auch beim Einschalten!

Grundsätzlich kann man sagen, daß der Rechner beim Einschalten vorzugsweise den C128-Modus initialisiert (logischerweise). Das wird schon anders, befindet sich eine CP/M-Diskette im Laufwerk. Sie haben ja bereits gelesen, daß der C128 beim Einschalten zunächst einmal eine Diskette im Laufwerk zu booten versucht. Befindet sich hier eine CP/M-Diskette, so wird halt die Z-80 eingeschaltet und die Kontrolle an dieselbe abgegeben (siehe auch Z-80-ROM). Aber auch über den BOOT-Befehl ist dies möglich.

Den C64-Modus erreichen Sie durch GO 64. Nach einer entsprechenden Sicherheitsabfrage wird der C64-Modus eingeschaltet. Weitere Möglichkeit: Sie haben eine Cartridge im entsprechenden Cartridge-Slot. Auch dann wird in den C64-Modus verzweigt.

Aber auch softwaremäßig gelangen Sie in den C64-Modus - ohne Sicherheitsabfrage. Dazu bedarf es eigentlich nur des folgenden Kommandos:

SYS 57931

Natürlich gibt es auch hier keine Rückkehr. Auch ist die Zehnertastatur im C64-Modus vollkommen blockiert, es gibt keine Möglichkeit, die zusätzlichen Tasten abzufragen.

Aber schon beim Einschalten können Sie auswählen, ob Sie nicht lieber in den C64-Modus wollen. Dies können Sie erreichen, indem Sie *direkt* beim Einschalten die Commodore-Taste drücken und gedrückt halten. Der Rechner versucht erst garnicht, einen Bootdurchgang durchzuführen. Übrigens funktioniert dies auch beim Betätigen der RESET-Taste.

Halten Sie entsprechend beim RESET die RUN/STOP-Taste gedrückt, so springt der Rechner zwar in den C128-Modus, aber sofort in den Maschinensprachemonitor. Für Maschinenprogrammierer also sehr interessant. Verlassen Sie den Maschinensprachemonitor mit dem X-Befehl, so wird erst *dann* der Bootvorgang durchgeführt.

Wir haben diese Möglichkeiten des Modusauswahl beim Einschalten zu schätzen gelernt - speziell, will man 64er-Programme fahren oder in Maschinensprache etwas programmieren. Will man beispielsweise ein Programm debuggen, daß direkt gebootet wird, so ist die Möglichkeit mit der RUN/STOP-Taste sogar die einzige bekannte, um *vorher* Werte im Speicher zu manipulieren.

Diese Abfragen können Sie übrigens besonders gut im dokumentierten Z-80-ROM-Listing nachverfolgen - auch wir sind erst beim Dokumentieren desselben auf die genannten Möglichkeiten gestoßen.

Kapitel 10: Tabellen und Stichwortverzeichnis

Die Register des VIC-Chip

Hex Adress	Decimal Adress	Offset	Funktion							
D000	53248	0	Sprite 0 X-Position (low 8 Bits)							
D001	53249	1	Sprite 0 Y-Position							
D002	53250	2	Sprite 1 X-Position (low 8 Bits)							
D003	53251	3	Sprite 1 Y-Position							
D004	53252	4	Sprite 2 X-Position (low 8 Bits)							
D005	53253	5	Sprite 2 Y-Position							
D006	53254	6	Sprite 3 X-Position (low 8 Bits)							
D007	53255	7	Sprite 3 Y-Position							
D008	53256	8	Sprite 4 X-Position (low 8 Bits)							
D009	53257	9	Sprite 4 Y-Position							
D00A	53258	10	Sprite 5 X-Position (low 8 Bits)							
D00B	53259	11	Sprite 5 Y-Position							
D00C	53260	12	Sprite 6 X-Position (low 8 Bits)							
D00D	53261	13	Sprite 6 Y-Position							
D00E	53262	14	Sprite 7 X-Position (low 8 Bits)							
D00F	53263	15	Sprite 7 Y-Position							
D010	53264	16	High Byte of Sprite X-Position							
			Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
D011	53265	17	Bit 8	Extended Color Mode 1 = on	Bitmap Mode 1 = on	Screen Blanking 0 = blank	No.of Rows 1 = 25 0 = 24	Vertical Screen Position 0 - 7 Pixels		
D012	53266	18	Raster Scan Line and write Register of Raster Interrupts							
			Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

D013	53267	19	Light Pen Horizontal Position							
D014	53268	20	Light Pen Vertical Position							
D015	53269	21	Enable Sprites (1 = on , 0 = off)							
			Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
D016	53270	22	1	1	Chip Reset 0 = Normal	Multicolor Mode 1 = Enable	No.of Columns 1=40,0=38	Horizontal Screen Position 0 - 7 Pixels		
D017	53271	23	Sprite Vertical Expansion (1 = on , 0 = off)							
			Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
D018	53272	24	Screen Base Adress				Character Definitions Base Adress			
			Bit 13	Bit 12	Bit 11	Bit 10	Bit 13	Bit 12	Bit 11	Bit 10
D019	53273	25	Interrupt Flags and write Register to clear Flags							
			Interrupts 1 = on 0 = off	1	1	1	Light Pen	Sprite - Sprite Collision	Sprite - Data Collision	Raster Scan
D01A	53274	26	Interrupt enable (1 = on , 0 = off)							
			1	1	1	1	Light Pen	Sprite - Sprite Collision	Sprite - Data Collision	Raster Scan
D01B	53275	27	Sprite Data Priority (1 = data , 0 = sprite)							
			Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
D01C	53276	28	Sprite Multicolor Mode (1 = enabled , 0 = not enabled)							
			Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
D01D	53277	29	Sprite Horizontal Expansion (1 = on , 0 = off)							
			Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
D01E	53278	30	Sprite - Sprite Collision Register (cleared only when read)							
			Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
D01F	53279	31	Sprite - Data Collision Register (cleared only when read)							
			Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0

Hex Adress	Decimal Adress	Offset	Funktion				
D020	53280	32	1	1	1	1	Border Color (0 - 15)
D021	53281	33	1	1	1	1	Background Color 0 (0 - 15)
D022	53282	34	1	1	1	1	Background Color 1 (0 - 15)
D023	53283	35	1	1	1	1	Background Color 2 (0 - 15)
D024	53284	36	1	1	1	1	Background Color 3 (0 - 15)
D025	53285	37	1	1	1	1	Sprite Multicolor 0
D026	53286	38	1	1	1	1	Sprite Multicolor 1
D027	53287	39	1	1	1	1	Sprite 0 Color
D028	53288	40	1	1	1	1	Sprite 1 Color
D029	53289	41	1	1	1	1	Sprite 2 Color
D02A	53290	42	1	1	1	1	Sprite 3 Color
D02B	53291	43	1	1	1	1	Sprite 4 Color
D02C	53292	44	1	1	1	1	Sprite 5 Color
D02D	53293	45	1	1	1	1	Sprite 6 Color
D02E	53294	46	1	1	1	1	Sprite 7 Color

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	BRK	BPL	JSR	BMI	RTI	BVC	RTS	BVS	ILL	BCC	LDY #	BCS	CPY #	BNE	CPX #	BEQ
1	ORA (,X)	ORA ().Y	AND (,X)	AND (),Y	EOR (,X)	EOR (),Y	ADC (,X)	ADC (),Y	STA (,X)	STA (),Y	LDA (,X)	LDA (),Y	CMP (,X)	CMP (),Y	SBC (,X)	SBC (),Y
2	ILL	ILL	ILL	ILL	ILL	ILL	ILL	ILL	ILL	ILL	LDX #	ILL	ILL	ILL	ILL	ILL
3	ILL	ILL	ILL	ILL	ILL	ILL	ILL	ILL	ILL	/	ILL	/	ILL	ILL	ILL	ILL
4	ILL	ILL	BIT ZP	ILL	ILL	ILL	ILL	ILL	STY ZP	STY ZP,X	LDY ZP	LDY ZP,X	CPY ZP	ILL	CPX ZP	ILL
5	ORA ZP	ORA ZP,X	AND ZP	AND ZP,X	EOR ZP	EOR ZP,X	ADC ZP	ADC ZP,X	STA ZP	STA ZP,X	LDA ZP	LDA ZP,X	CMP ZP	CMP ZP,X	SBC ZP	SBC ZP,X
6	ASL ZP	ASL ZP,X	ROL ZP	ROL ZP,X	LSR ZP	LSR ZP,X	ROR ZP	ROR ZP,X	STX ZP	STX ZP,Y	LDX ZP	LDX ZP,Y	DEC ZP	DEC ZP,X	INC ZP	INC ZP,X
7	ILL	ILL	ILL	ILL	ILL	ILL	ILL	ILL	ILL	ILL	ILL	/	ILL	ILL	ILL	ILL
8	PHP	CLC	PLP	SEC	PHA	CLI	PLA	SEI	DEY	TYA	TAY	CLV	INY	CLD	INX	SED
9	ORA #	ORA ,Y	AND #	AND ,Y	EOR #	EOR ,Y	ADC #	ADC ,Y	ILL	STA ,Y	LDA #	LDA ,Y	CMP #	CMP ,Y	SBC #	SBC ,Y
A	ASL AKKU	ILL	ROL AKKU	ILL	LSR AKKU	ILL	ROR AKKU	ILL	TXA	TXS	TAX	TSX	DEX	ILL	NOP	ILL
B	ILL	ILL	ILL	ILL	ILL	ILL	/	ILL	/	ILL	ILL	ILL	ILL	ILL	ILL	ILL
C	ILL	ILL	BIT	ILL	JMP	ILL	JMP (),Y	ILL	STY	ILL	LDY	LDY ,X	CPY	ILL	CPX	ILL
D	ORA	ORA ,X	AND	AND ,X	EOR	EOR ,X	ADC	ADC ,X	STA	STA ,X	LDA	LDA ,X	CMP	CMP ,X	SBC	SBC ,X
E	ASL	ASL ,X	ROL	ROL ,X	LSR	LSR ,X	ROR	ROR ,X	STX	ILL	LDX	LDX ,Y	DEC	DEC ,X	INC	INC ,X
F	ILL	ILL	ILL	ILL	ILL	ILL	ILL	ILL	/	ILL	ILL	ILL	ILL	ILL	ILL	ILL

	IMM	ABS	ABS ,X	ABS ,Y	ZP	ZP ,X	ZP ,Y	(,X)	( ),Y	REL	IND	AKKU	IMPL
CPX	E0 2	EC 4			E4 3								
CPY	C0 2	CC 4			C4 3								
BIT		2C 4			24 3								
BCC										90 2*			
BCS										B0 2*			
BEQ										F0 2*			
BNE										D0 2*			
BMI										30 2*			
BPL										10 2*			
BVC										50 2*			
BVS										70 2*			
JMP		4C 3									6C 5		
JSR		20 6											
ASL		0E 6	1E 7		06 5	16 6						0A 2	
LSR		4E 6	5E 7		46 5	56 6						4A 2	
ROL		2E 6	3E 7		26 5	36 6						2A 2	
ROR		6E 6	7E 7		66 5	76 6						6A 2	
CLC													18 2
CLD													D8 2
CLI													58 2
CLV													B8 2
SEC													38 2
SED													F8 2
SEI													78 2
NOP													EA 2
RTS													60 6
RTI													40 6
BRK													00 7

A0 03

1.Zahl = Opcode

2.Zahl = Taktzyklen pro Befehl

(* = bei Bereichsüberschreitung 1 Taktzyklus mehr)
(Alle BRANCH - Befehle + 1 Taktzyklus, wenn Bedingung zutrifft)

	IMM	ABS	ABS ,X	ABS ,Y	ZP	ZP ,X	ZP ,Y	(,X)	( ),Y	REL	IND	AKKU	IMPL
LDA	A9 2	AD 4	BD 4*	B9 4*	A5 3	B5 4		A1 6	B1 5*				
LDX	A2 2	AE 4		BE 4*	A6 3		B6 4						
LDY	A0 2	AC 4	BC 4		A4 3	B4 4*							
STA		8D 4	9D 5	99 5	85 3	95 4		81 6	91 6				
STX		8E 4			86 3		96 4						
STY		8C 4			84 3	94 4							
TAX													AA 2
TAY													A8 2
TXA													8A 2
TYA													98 2
TXS													9A 2
TSX													BA 2
PLA													68 4
PHA													48 3
PLP													28 4
PHP													08 3
ADC	69 2	6D 4	7D 4*	79 4*	65 3	75 4		61 6	71 5*				
SBC	E9 2	ED 4	FD 4*	F9 4*	E5 3	F5 4		E1 6	F1 5*				
INC		EE 6	FE 7		E6 5	F6 6							
DEC		CE 6	DE 7		C6 5	D6 6							
INX													E8 2
DEX													CA 2
INY													C8 2
DEY													88 2
AND	29 2	2D 4	3D 4*	39 4*	25 3	35 4		21 6	31 5*				
ORA	09 2	0D 4	1D 4*	19 4*	05 3	15 4		01 6	11 5*				
EOR	49 2	4D 4	5D 4*	59 4*	45 3	55 4		41 6	51 5*				
CMP	C9 2	CD 4	DD 4*	D9 4*	C5 3	D5 4		C1 6	D1 5*				

A0 03

1.Zahl = Opcode

2.Zahl = Taktzyklen pro Befehl

(* = bei Bereichsüberschreitung 1 Taktzyklus mehr)
(Alle BRANCH – Befehle + 1 Taktzyklus, wenn Bedingung zutrifft)

Dezi	Hex	Binär	Dezi	Hex	Binär
#000	$00	%00000000	#001	$01	%00000001
#002	$02	%00000010	#003	$03	%00000011
#004	$04	%00000100	#005	$05	%00000101
#006	$06	%00000110	#007	$07	%00000111
#008	$08	%00001000	#009	$09	%00001001
#010	$0A	%00001010	#011	$0B	%00001011
#012	$0C	%00001100	#013	$0D	%00001101
#014	$0E	%00001110	#015	$0F	%00001111
#016	$10	%00010000	#017	$11	%00010001
#018	$12	%00010010	#019	$13	%00010011
#020	$14	%00010100	#021	$15	%00010101
#022	$16	%00010110	#023	$17	%00010111
#024	$18	%00011000	#025	$19	%00011001
#026	$1A	%00011010	#027	$1B	%00011011
#028	$1C	%00011100	#029	$1D	%00011101
#030	$1E	%00011110	#031	$1F	%00011111
#032	$20	%00100000	#033	$21	%00100001
#034	$22	%00100010	#035	$23	%00100011
#036	$24	%00100100	#037	$25	%00100101
#038	$26	%00100110	#039	$27	%00100111
#040	$28	%00101000	#041	$29	%00101001
#042	$2A	%00101010	#043	$2B	%00101011
#044	$2C	%00101100	#045	$2D	%00101101
#046	$2E	%00101110	#047	$2F	%00101111
#048	$30	%00110000	#049	$31	%00110001
#050	$32	%00110010	#051	$33	%00110011
#052	$34	%00110100	#053	$35	%00110101
#054	$36	%00110110	#055	$37	%00110111
#056	$38	%00111000	#057	$39	%00111001
#058	$3A	%00111010	#059	$3B	%00111011
#060	$3C	%00111100	#061	$3D	%00111101
#062	$3E	%00111110	#063	$3F	%00111111
#064	$40	%01000000	#065	$41	%01000001
#066	$42	%01000010	#067	$43	%01000011
#068	$44	%01000100	#069	$45	%01000101
#070	$46	%01000110	#071	$47	%01000111
#072	$48	%01001000	#073	$49	%01001001
#074	$4A	%01001010	#075	$4B	%01001011
#076	$4C	%01001100	#077	$4D	%01001101
#078	$4E	%01001110	#079	$4F	%01001111
#080	$50	%01010000	#081	$51	%01010001
#082	$52	%01010010	#083	$53	%01010011
#084	$54	%01010100	#085	$55	%01010101

Dezi	Hex	Binär	Dezi	Hex	Binär
----	---	---------	----	---	---------
#086	$56	%01010110	#087	$57	%01010111
#088	$58	%01011000	#089	$59	%01011001
#090	$5A	%01011010	#091	$5B	%01011011
#092	$5C	%01011100	#093	$5D	%01011101
#094	$5E	%01011110	#095	$5F	%01011111
#096	$60	%01100000	#097	$61	%01100001
#098	$62	%01100010	#099	$63	%01100011
#100	$64	%01100100	#101	$65	%01100101
#102	$66	%01100110	#103	$67	%01100111
#104	$68	%01101000	#105	$69	%01101001
#106	$6A	%01101010	#107	$6B	%01101011
#108	$6C	%01101100	#109	$6D	%01101101
#110	$6E	%01101110	#111	$6F	%01101111
#112	$70	%01110000	#113	$71	%01110001
#114	$72	%01110010	#115	$73	%01110011
#116	$74	%01110100	#117	$75	%01110101
#118	$76	%01110110	#119	$77	%01110111
#120	$78	%01111000	#121	$79	%01111001
#122	$7A	%01111010	#123	$7B	%01111011
#124	$7C	%01111100	#125	$7D	%01111101
#126	$7E	%01111110	#127	$7F	%01111111
#128	$80	%10000000	#129	$81	%10000001
#130	$82	%10000010	#131	$83	%10000011
#132	$84	%10000100	#133	$85	%10000101
#134	$86	%10000110	#135	$87	%10000111
#136	$88	%10001000	#137	$89	%10001001
#138	$8A	%10001010	#139	$8B	%10001011
#140	$8C	%10001100	#141	$8D	%10001101
#142	$8E	%10001110	#143	$8F	%10001111
#144	$90	%10010000	#145	$91	%10010001
#146	$92	%10010010	#147	$93	%10010011
#148	$94	%10010100	#149	$95	%10010101
#150	$96	%10010110	#151	$97	%10010111
#152	$98	%10011000	#153	$99	%10011001
#154	$9A	%10011010	#155	$9B	%10011011
#156	$9C	%10011100	#157	$9D	%10011101
#158	$9E	%10011110	#159	$9F	%10011111
#160	$A0	%10100000	#161	$A1	%10100001
#162	$A2	%10100010	#163	$A3	%10100011
#164	$A4	%10100100	#165	$A5	%10100101
#166	$A6	%10100110	#167	$A7	%10100111
#168	$A8	%10101000	#169	$A9	%10101001
#170	$AA	%10101010	#171	$AB	%10101011
#172	$AC	%10101100	#173	$AD	%10101101

Dezi	Hex	Binär	Dezi	Hex	Binär
----	---	---------	----	---	---------
#174	$AE	%10101110	#175	$AF	%10101111
#176	$B0	%10110000	#177	$B1	%10110001
#178	$B2	%10110010	#179	$B3	%10110011
#180	$B4	%10110100	#181	$B5	%10110101
#182	$B6	%10110110	#183	$B7	%10110111
#184	$B8	%10111000	#185	$B9	%10111001
#186	$BA	%10111010	#187	$BB	%10111011
#188	$BC	%10111100	#189	$BD	%10111101
#190	$BE	%10111110	#191	$BF	%10111111
#192	$C0	%11000000	#193	$C1	%11000001
#194	$C2	%11000010	#195	$C3	%11000011
#196	$C4	%11000100	#197	$C5	%11000101
#198	$C6	%11000110	#199	$C7	%11000111
#200	$C8	%11001000	#201	$C9	%11001001
#202	$CA	%11001010	#203	$CB	%11001011
#204	$CC	%11001100	#205	$CD	%11001101
#206	$CE	%11001110	#207	$CF	%11001111
#208	$D0	%11010000	#209	$D1	%11010001
#210	$D2	%11010010	#211	$D3	%11010011
#212	$D4	%11010100	#213	$D5	%11010101
#214	$D6	%11010110	#215	$D7	%11010111
#216	$D8	%11011000	#217	$D9	%11011001
#218	$DA	%11011010	#219	$DB	%11011011
#220	$DC	%11011100	#221	$DD	%11011101
#222	$DE	%11011110	#223	$DF	%11011111
#224	$E0	%11100000	#225	$E1	%11100001
#226	$E2	%11100010	#227	$E3	%11100011
#228	$E4	%11100100	#229	$E5	%11100101
#230	$E6	%11100110	#231	$E7	%11100111
#232	$E8	%11101000	#233	$E9	%11101001
#234	$EA	%11101010	#235	$EB	%11101011
#236	$EC	%11101100	#237	$ED	%11101101
#238	$EE	%11101110	#239	$EF	%11101111
#240	$F0	%11110000	#241	$F1	%11110001
#242	$F2	%11110010	#243	$F3	%11110011
#244	$F4	%11110100	#245	$F5	%11110101
#246	$F6	%11110110	#247	$F7	%11110111
#248	$F8	%11111000	#249	$F9	%11111001
#250	$FA	%11111010	#251	$FB	%11111011
#252	$FC	%11111100	#253	$FD	%11111101
#254	$FE	%11111110	#255	$FF	%11111111

User Port



PIN	SIGNAL	BEMERKUNG
1	GND	
2	+5V	MAX. 100 mA
3	$\overline{\text{RESET}}$	
4	CNT1	
5	SP1	
6	CNT2	
7	SP2	
8	$\overline{\text{PC2}}$	
9	SER. ATN IN	
10	9 VAC	MAX. 100 mA
11	9 VAC	MAX. 100 mA
12	GND	

PIN	SIGNAL
A	GND
B	$\overline{\text{FLAG 2}}$
C	PB 0
D	PB 1
E	PB 2
F	PB 3
H	PB 4
J	PB 5
K	PB 6
L	PB 7
M	PA 2
N	GND

22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Z	Y	X	W	V	U	T	S	R	P	N	M	L	K	J	H	F	E	D	C	B	A

Pin	Signal
1	GND
2	+ 5V
3	+ 5V
4	$\overline{IRQ}$
5	CR/$\overline{W}$
6	DOT CLOCK
7	I/0 1
8	$\overline{GAME}$
9	$\overline{EXROM}$
10	I/0 2
11	ROML
12	BA
13	$\overline{DMA}$
14	D7
15	D6
16	D5
17	D4
18	D3
19	D2
20	D1
21	D0
22	GND
A	GND
B	ROMH
C	$\overline{RESET}$
D	NMI
E	Φ 2
F	A15
H	A14
J	A13
K	A12
L	A11
M	A10
N	A9
P	A8
R	A7
S	A6
T	A5
U	A4
V	A3
W	A2
X	A1
Y	A0
Z	GND

NR.	NOTE – OKTAVE	FREQUENZ	HIGH – BYTE	LOW – BYTE
1	C – 0	16.4	1 (01)	22 (16)
2	C# – 0	17.3	1 (01)	39 (27)
3	D – 0	18.4	1 (01)	57 (39)
4	D# – 0	19.4	1 (01)	75 (4B)
5	E – 0	20.6	1 (01)	95 (5F)
6	F – 0	21.8	1 (01)	116 (74)
7	F# – 0	23.1	1 (01)	138 (8A)
8	G – 0	24.5	1 (01)	161 (A1)
9	G# – 0	26	1 (01)	186 (BA)
10	A – 0	27.5	1 (01)	212 (D4)
11	A# – 0	29.1	1 (01)	240 (F0)
12	H – 0	30.9	2 (02)	14 (0E)
13	C – 1	32.7	2 (02)	45 (2D)
14	C# – 1	34.6	2 (02)	78 (4E)
15	D – 1	36.7	2 (02)	113 (71)
16	D# – 1	38.9	2 (02)	150 (96)
17	E – 1	41.2	2 (02)	190 (BE)
18	F – 1	43.7	2 (02)	231 (E7)
19	F# – 1	46.2	3 (03)	20 (14)
20	G – 1	49.0	3 (03)	66 (42)
21	G# – 1	51.9	3 (03)	116 (74)
22	A – 1	55.0	3 (03)	169 (A9)
23	A# – 1	58.3	3 (03)	224 (E0)
24	H – 1	61.7	4 (04)	27 (1B)
25	C – 2	65.4	4 (04)	90 (5A)
26	C# – 2	69.3	4 (04)	156 (9C)
27	D – 2	73.4	4 (04)	226 (E2)
28	D# – 2	77.8	5 (05)	45 (2D)
29	E – 2	82.4	5 (05)	123 (7B)
30	F – 2	87.3	5 (05)	207 (CF)
31	F# – 2	92.5	6 (06)	39 (27)
32	G – 2	98.0	6 (06)	133 (85)
33	G# – 2	103.8	6 (06)	232 (E8)
34	A – 2	110.0	7 (07)	81 (51)
35	A# – 2	116.5	7 (07)	193 (C1)
36	H – 2	123.5	8 (08)	55 (37)
37	C – 3	130.8	8 (08)	180 (B4)
38	C# – 3	138.6	9 (09)	56 (38)
39	D – 3	146.8	9 (09)	196 (C4)
40	D# – 3	155.6	10 (0A)	89 (59)
41	E – 3	164.8	10 (0A)	247 (F4)
42	F – 3	174.6	11 (0B)	158 (9E)
43	F# – 3	185.0	12 (0C)	78 (4E)
44	G – 3	196.0	13 (0D)	10 (0A)
45	G# – 3	207.7	13 (0D)	208 (D0)
46	A – 3	220.0	14 (0E)	162 (A2)
47	A# – 3	233.1	15 (0F)	129 (81)

NR.	NOTE – OKTAVE	FREQUENZ	HIGH – BYTE	LOW – BYTE
48	H – 3	246.9	16 (10)	109 (6D)
49	C – 4	261.6	17 (11)	103 (67)
50	C# – 4	277.2	18 (12)	112 (70)
51	D – 4	293.7	19 (13)	137 (89)
52	D# – 4	311.1	20 (14)	178 (B2)
53	E – 4	329.6	21 (15)	237 (ED)
54	F – 4	349.2	23 (17)	59 (3B)
55	F# – 4	370.0	24 (19)	157 (9D)
56	G – 4	392.0	26 (1A)	20 (14)
57	G# – 4	415.3	27 (1B)	160 (A0)
58	A – 4	440.0	29 (1D)	69 (45)
59	A# – 4	466.2	31 (1F)	3 (03)
60	H – 4	493.9	32 (20)	219 (DB)
61	C – 5	523.3	34 (22)	207 (CF)
62	C# – 5	554.4	36 (24)	225 (E1)
63	D – 5	587.3	39 (27)	18 (12)
64	D# – 5	622.3	41 (29)	101 (65)
65	E – 5	659.3	43 (2B)	219 (DB)
66	F – 5	689.5	46 (2E)	118 (76)
67	F# – 5	740.0	49 (31)	58 (3A)
68	G – 5	784.0	52 (34)	39 (27)
69	G# – 5	830.6	55 (37)	65 (38)
70	A – 5	880.0	58 (3A)	138 (8A)
71	A# – 5	932.3	62 (3E)	5 (05)
72	H – 5	987.8	65 (41)	181 (B5)
73	C – 6	1046.5	69 (45)	157 (9D)
74	C# – 6	1108.7	73 (49)	193 (C1)
75	D – 6	1174.7	78 (4E)	36 (24)
76	D# – 6	1244.5	82 (52)	201 (C9)
77	E – 6	1318.5	87 (57)	182 (B6)
78	F – 6	1396.9	92 (5C)	237 (ED)
79	F# – 6	1480.0	98 (62)	115 (73)
80	G – 6	1568.0	104 (68)	78 (4E)
81	G# – 6	1661.2	110 (6E)	130 (82)
82	A – 6	1760.0	117 (75)	20 (14)
83	A# – 6	1864.7	124 (7C)	10 (0A)
84	H – 6	1975.5	131 (83)	106 (6A)
85	C – 7	2093.0	139 (8B)	59 (3B)
86	C# – 7	2217.5	147 (93)	130 (82)
87	D – 7	2349.3	156 (9C)	72 (48)
88	D# – 7	2489.0	165 (A5)	147 (93)
89	E – 7	2637.0	175 (AF)	107 (6B)
90	F – 7	2793.8	185 (B9)	218 (DA)
91	F# – 7	2960.0	196 (C4)	231 (E7)
92	G – 7	3136.0	208 (D0)	156 (9C)
93	G# – 7	3322.4	221 (DD)	4 (04)
94	A – 7	3520.0	234 (EA)	40 (28)
95	A# – 7	3729.3	248 (F8)	20 (14)

Stichwortverzeichnis

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

Z